NASA TECHNICAL MEMORANDUM NASA TM-88528

DESIGN OF SOFTWARE FOR DESIGN OF FINITE ELEMENT FOR STRUCTURAL ANALYSIS

Reinhard Helfrich

Translation of: "Zur Entwicklung eines Softwaresystems fur die Modellbeschreibung bei der Methode der finiten Elemente," dissertation submitted to the Faculty of Aerospace Transportation Technology of the University of Stuttgart for the Achievement of the Degree of Doctor of Engineering, (Nov. 22) 1983, pp. 1-192

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D.C. 20546 FEBRUARY 1987

Toward the Development of a Software System for Modelling with the Finite Element Method

from the Faculty of Aerospace Transportation Technology
of the University of Stuttgart for the Achievement of the Degree of
Doctor of Engineering (Dr.-Ing.)

Submitted by
Reinhard Helfrich
from Ludwigsburg

Advisor:	Prof.Dr.Dr.h.c.mult. J. H. Argyris
Referee:	Prof. W. Schönauer
Date Submitted:	22 November 1983
Date of Oral Examination:	25 July 1984

Institute for Statics and Dynamics of Aerospace Structures
of the University of Stuttgart

1984

Abstract

HELFRICH, REINHARD

TOWARD THE DEVELOPMENT OF A SOFTWARE SYSTEM FOR MODELLING WITH THE FINITE ELEMENT METHOD

This work is concerned with the concepts of software engineering which allow a user of the finite element method to describe his model, to collect and to check the model data in a data base as well as to form the matrices required for a finite element calculation. Next the components of the model description are conceived including the mesh tree, the topology, the configuration, the kinematic boundary conditions, the data for each element and the loads. For this the possibilities for description and review of the data are especially considered. The concept of the segments for the modularization of the programs follows the components of the model description. The significance of the mesh tree as a global guiding structure will be understood in view of the principle of the unity of the model, mesh tree, and the database. Out of this will be derived the concept of the mesh context for the modularization of the data in the data base. For storage of the model data, the data module sequences and tables will be introduced and their application functions thoroughly described. Then the user interface will be developed through the central concept of command language. It embraces the recognition and description of the language, the production of language tables and with that, the possible translation of every language input in free format to a fixed processing format for the programs. At the same time the concepts of open lists for the creation of regular series of numbers and macro-commands for user-specific definition and application of repeatable and variable parts of the modelling will be explained/ Finally the user-friendly aspects of the software system will be summarized: the principle of language communication, the data generators, error processing, and data security.

*Numbers in margin indicate foreign pagination.

Foreword

The following work is based on the research activities of the ISD (Insitute for Statics and Dynamics) in the framework of the further development of the finite element programming system ASKA, in which work the author was allowed to participate. Professor Dr. J. H. Argyris is to be thanked that the work could be performed under such favorable professional and humane conditions. The author especially wishes to thank cordially Professor Argyris for his support.

Professor W. Schönauer is cordially thanked as well for his support of the work and for taking over the duties of referee.

Also thanked are all the colleagues at the ISD, especially Dr. Ernst Schrem, who stimulated the work in numerous discussions and Mr. Uwe Schulz, whose constant collaboration enabled the implementation of the first experimental program.

My wife, Brigitte Helfrich, earns special thanks for her encouraging solidarity and the painstaking typing of the manuscript.

Table of Contents

		Page
	Explanation of Symbols	8
1.	Introduction	10
2.	General Aspects of System Design	14
	2.1 Core Systems and Satellite Systems	14
	2.2 System and Functional Unity	15
	2.3 Application and Development Environment	18
3.	Fundamentals and Components of Modelling	22
	3.1 Brief Characterization	22
	3.2 Element and Mesh	23
	3.3 Mesh Tree	27
	3.4 Nodes and Incidences	31
	3.5 Topology	33
	3.5.1 Description	33
	3.5.2 Degree of Incidence	34
	3.5.3 Euler-Poincaré Characteristic	37
	3.6 Configuration	42
	3.6.1 Rules of Description	42
	3.6.2 Coordinates of Nodes	43
	3.6.3 Regular Description of Nodes	49
	3.6.4 Node Basis	55
	3.7 Duality of the Degree of Freedom	62
	3.7.1 Dual Vector Spaces in Physics	62
	3.7.2 Duality Relationships	64
	3.7.3 Displacement Methods	67
	3.8 Kinematic Boundary Conditions	68
	3.8.1 Incidences of Degrees of Freedom	68
	3.8.2 Classes of Degrees of Freedom	70
	3.8.3 Element Degrees of Freedom	74

3.9 Element Data 75

3.10 Loading 78

3.11 Calculation 82

4. Modularization of the Program and the Data 84

4.1 Segments 84
4.1.1 Concepts 84
4.1.2 Control 88

4.2 Contexts 92
4.2.1 Concepts 92
4.2.2 Mesh Lists 96
4.2.3 Traverses of Mesh Tree 98
4.2.4 Control of the Mesh Context 99

/5

5. Management of the Model Data 107

5.1 General Aspects of Memory Management 107

5.2 Control Lists 110
5.2.1 Types 110
5.2.2 Structures 112
5.2.3 System Description 119

5.3 Model Data 119
5.3.1 Numbering 119
5.3.2 Tables 121
5.3.3 Project Maintenance 125

6. Command Language 128

6.1 Overview 128

6.2 Lexical Analysis and Free Format 133

6.3 Syntax 136

6.4 Aspects of Language Processing 140
6.4.1 Language Recognition 140
6.4.2 Graphic Depiction of Syntax 142
6.4.3 Language Description 144
6.4.4 Language Creation 149
6.4.5 Language Translation 152
6.4.6 Functions for Language Translation 155

6.5 Open Lists 158
6.5.1 Concept 158
6.5.2 Functions for List Creation 159
6.5.3 Description of Lists 163

6.6 Macro Commands 171
6.6.1 Properties 171
6.6.2 Macro Description 173

7. Aspects of User-Friendly Modelling 180

7.1 Principles of Communication 180

7.2 Data Generators 182

7.3 Errors 185

7.4 Data Security 191

Literature 193

List of Figures 200

Appendix A: Example of Modelling 203
B: Linear Static Analysis 209
C: Proof of Eq. (3.15) 212

Explanation of Symbols

1. Ways of Writing Matrices

$\mathbf{a}, \mathbf{A}$	: Vectors and matrices are boldface
$[a_i]$	: Column matrix with elements a_i
$\lceil a_i \rfloor$	: Diagonal matrix with elements a_i
$\{a_1 \cdot\cdot a_n\}$	: Column Matrix with elements a_1 to a_n in row order

2. Ways of Writing Brackets for Description of Commands

α, β	: Parts of commands. If they are not enclosed within angle brackets, they must be used in the given form.
$\left\{\begin{matrix}\alpha \\ \beta\end{matrix}\right\}$	: α or β must be written
$\{\alpha\}^*, \{\alpha\}^m, \{\alpha\}_n^m$	: α is to be written at least once (or n times, respectively and repeated as often as desired (or m times, repectively.
$\left[\begin{matrix}\alpha \\ \beta\end{matrix}\right]$	: α or β can be written
$[\alpha]^*, [\alpha]^m, [\alpha]_n^m$	: If α is written, then it must be written at least once (or n times, respectively) and α must be repeated as often as desired (or a maximum of m times).
$\{[\alpha]\ [\beta]\}$	: At least one of command parts α and β must be written

<a> : All intermediate symbols of the command description will be written lowercase between angle brackets. During the use of the command, these intermediate symbols are to be replaced by the user with suitable end symbols.

Other possibilities for describing the commands are found in Chapter 6.4.2.

3. Function Description

AAA (a_1, .. , a_i, $\underline{a}_j$, ..)

AAA : Function name in uppercase letters

a_i, a_j : Function parameter in lowercase letters. They depict intermediate symbols which are to be replaced by the programmer with end symbols (e.g. with FORTRAN variable names.

a_i : Input parameter

$\underline{a}_j$: Output parameter. These are emphasized by underlining.

$a_i[d]$: The parameter a_i stands for several pieces of data, so it is to be placed in a field of length d.

$a_i[d_1, d_2]$: Two dimensional field with column dimension d_i and row dimension d_2

4. Operations on integers

div (m,n) : integer quotient of the division of m through n.

mod (m,n) : The remainder from the division div (m,n)

1. Introduction

The method of finite elements (FE) is a recognized procedure of applied mathematics. Since the beginning of its development [Argyris 54,56,57] it has been intimately connected with the technology of digital computers. The early insight into the capabilities of this then-new technology has significantly influenced the development of the method in form of the displacement method [Argyris 59,64,65,70,75]. Therefore it was logical to connect the creation of the theoretical fundamentals to the development of a powerful programming system, which could serve a broad application of the method. [Schrem, 70a,70b,71,75]

Since the introduction of the first programming system suited for the finite element method, the conditions for its applications have changed in manifold aspects. If, at the beginning, structural calculations were established as an area of application, so today many areas of application have been opened up. If at first only large research establishments and financially strong industrial concerns were in a position to buy a digital computer, the development of inexpensive and small computers has advantageously changed the prerequisites for the application of the method. Out of this resulted a constantly increasing dissemination of the method and the software systems required for its practical application.

Software systems, as all technical creations, are first objects of the development and afterwards of the application. Thus those who develop the system and those who apply the system, the so-called users, are commonly not identical. Therefore the users play a decisive role in all development considerations, This applies above all for software systems for the finite element method, because its scope and complexity require considerable development costs and the broad dissemination which is therefore necessary presupposes the approval of many users. The judgement of a user will be determined on

one hand by the fulfillment of functional requirements, on the other hand through the user's experience in working with a system. Out of this results the importance of the so-called user interface, over which the entire communication between user and system is developed. A user interface is then called user-friendly if it connects /9 numerous functions to reduce work with a pleasant unified way of working. In particular, the modelling is a very communication--intensive phase in which a slight or a well-developed user-friendliness of the interface easily becomes evident.

At the beginning of the history of the development of software systems for the finite element method the user interface was not given the attention necessary today. At that time one concentrated more on the algorithmic aspects of FE calculation. That was also required from the state of the hardware and software equipment, since only modest resources were available in terms of computing time and storage capacity. With the improvement of these resources it was recognized that the preparation and input of the model data as well as the evaluation of the results together required 60 to 90 percent of the total time needed for application of the method (after [Gallagher 77, Rehak 81]). A shortening of this time is thus to be achieved through a reevaluation of the user interface in view of earlier system solutions. Thereby the user-friendliness of the interface comes into decisive significance. How this can be achieved shall be depicted in the following work in terms of modelling.

The actual computation programs apply the method using matrices. It was always obvious that it could not be demanded of every user to construct the matrices himself and place the program at his disposal. Therefore the job of a software system for modelling is to place in readiness tools which allow the user to describe a finite element model in terms amenable to him (such as location of elements, orientation of the supporting structure, material properties), to collect and to check the model data as well as to create the matrices necessary for the calculation.

Tools of the type described are made available by practically all software systems for the method of finite elements. There, supplying the system with data is in the foreground and not the user friendliness of the data entry. Besides that, many pre-run programs (so-called pre-processors have been developed which support a user- or application-oriented model description. However, all too often /10 the description of a model is equated with the pure division into elements and establishment of coordinates (keyword: mesh generators). Therefore, in the following work it will first be thoroughly examined which components a model description does have. Special prerequisites for this were the application of the partial structure process, which is also applied to the model calculations and the incorporation of the element calculations into the model description in accordance with the above-mentioned assignments. In no case was it intended to develop a general multipurpose program that would be suitable as a pre-processor for all FE systems. The development had rather the goal to merge the model description into the virtual FE machine (VFE machine) newly conceived by Schrem [Schrem 78a]. Incidentally, in other respects in this work expressions and examples from structural calculations will be cited where necessary for explanation.

A complete model description requires in some circumstances that the user make available very extensive amounts of data. A user-friendly system must therefore realize concepts for the simplified and abbreviated description of the data. The wish of the user to describe ever-more complex models with ever-smaller amounts of data is understandable. Precisely for that reason, all questions which are connected with the verification of the correctness of a model will be thoroughly treated in this work. Because as long as software systems don't operate according to the utopian demands of the users: "I'm thinking of a model, make it available!", incompatibilities and errors will belong to the normal byproducts of a model description. For so long, the user will have to communicate his model data to the system in some manner. To create a foundation for

this in the form of a command language is a further chief concern of this work. Here is appplied extensive experience from applied information science with the help of which a language concept suited for the model description will be developed in a very general way. Special attributes for this include on one hand the description of number series for the creation of desired model data and on the other hand the capability for the depiction of algorithms with which the user can realize variable model descriptions.

/11

Software is designated above all by the category of information and not by the category of material or energy. Therefore one cannot get to know software by handling it or looking at it. This results in difficulties for the depiction of a software outline, for which there presently exists no satisfactory method (as with blueprints, for example). The following work therefore attempts to depict the outline of the software system under three different aspects. For the first will be explained the decisions for the outline which are applicable for system configuration. In addition above all will be emphasized the observed principles whose effectiveness marks a software system of the level expected of a technical product of high quality. The principle of unity of model, mesh tree and project data base proves itself to be of particular far-reaching significance. Second, the central functional units will be described. Their functions and data structures will be described as immediate consequences of abstract concepts. The function description serves as a basis for the implementation of the functional unity. The complete software system will be implemented through the assembly of equally-entitled and hierarchically-ordered fuctional units. Third, the commands for the user will be introduced. Through them, the functions and capabilities of the user inter face will be stipulated. They facilitate a presentation of the practice of the use of the system in textual connection with the underlying concepts and principals. Above all, with respect to the strived-for user friendliness, the commands are to be seen as significant results of the system development.

2. General Aspects of the System Outline

2.1 Core Systems and Satellite Systems

A software system for the method of finite elements will be decisively influenced by the versatility and uniformity of the method. On one hand the versatility is represented by the element model which allows the most diverse areas of application to be comprehended. On the other hand the unity of the method is guaranteed through the operations of linear algebra, which make possible the formulation and solution of the systems of equations arising from the discretization. The suitability of this method was already recognized in the beginning of its development: "The matrix formulation allows us not only to write the equations much more clearly, rather it is also the ideal way of writing for digital computers" [Argyris 57].

The separation of element model and solution of equations is reflected in the structuring of the software system. Thus, the unity of the operation of linear algebra has led to the concept of the core system, in which abstractions can be made from all element models. So, for example, the following functions of elastic static mechanics do not belong in the core system:

- o assembly of the element stiffness matrices,
- o determination of initial loads based on initial strains,
- o discretization of distributed element loads (such as line, area and volume loads),
- o calculation of element stresses from element strains.

Through the exclusion of these functions the core system becomes very generally applicable and its application is exclusively oriented to the solution algorithm.

The concept complementary to the core system is that of the

satellite system that, together with the above-mentioned element-dependent functions comprises the overall model description and evaluation of results. With this,all application-dependent and communication-intensive parts of a FE calculation are clearly divided from the core system. A change in the area of application can subsequently lead to a new satellite system, while the core system with all computation-intensive parts remains unchanged. The /13
integration of core system and satellite system is designated virtual finite element machine (VFE machine), "which is suited as fundamental ordering principle for every modern programming system for the application of the method of finite elements" [Schrem 78a]. The two partial systems are connected with each other over the data-processing system (DVS) {Datenverwaltungssystem} which constitutes an example of the VFE machine.

2.2 System and Functional Unity

The idea of the system was already used in this work in some expresions: software system, program system, core system, satellite system, data processing system. The idea should be examined here somwhat more closely in order to avoid its undisciplined use. In [Rohpohl 79] one finds the following description of the system idea: "a system is a unit which (a) exhibits relationships between definite attributes, which (b) consists of interconnected parts or subsystems, and (c) is divided by a definite boundary from its environment or is circumscribed within a supersystem." The functional view (a) describes a system by a number of attributes -- those are, for the time being, arbitrary properties. Such attributes, which place the system in relation to its environment, are called entrance and exit attributes. Such attributes which characterize the constitution of the system itself are called state attributes. Every coordination /14
of the attributes to each other represents a function of the system. This conception of the function is universal and signifies no limitation to mathematical functions. According to the structural

view (b) a system consists of subsystems which combine in a definite manner to form the whole. A system is therefore the environment of all its subsystems. A structure exists if definite relations exist between the subsystems. With these relations, interconnections between attributes of various subsystems are designated. If the subsystems arise through division of the entire system, one speaks of the modularization of the system; if a system arises from the assembly of subsystems, then the system exists as a result of integration. According to the hierarchical view (c) subsystems are further divisible into subsystems. Thus, a recursive process is described to modularize a system into smaller and smaller subsystems or to integrate systems into more and more powerful supersystems.

Software systems can now be described as technical systems for which all attributes are represented by data. These systems are technical because they are, on one hand, artificially created and, on the other hand serve as tools for achievement of definite tasks. For the subsystems of a software system a sound concept was introduced in [Schrem 78a,78b] with the idea of functional unity: " A functional unit is a device for data processing that is defined by a complete set of function and performance specifications" [Schrem 78a] (Emphasis added). As attributes of a functional unit the data may be divided into

- o input data: these are transmitted from the environment to the functional unit.
- o output data: these are transmitted from the functional unit to the environment.
- o internal data: these are known only within the functional unit and represent the state of the functional unit.

The function specification describes the environment of the /15
functional unit, the domain and the significance of all input and

output data, all complete functions as well as the changes of state connected with them. For the outline of a functional unit the following rules apply:

- o The function specification should be complete and should include all functionswhich result from the application of the functional unit. This also requires that for every input datum (or output datum or internal datum) there exists at least one function which needs (or creates or changes) the datum.
- o The function specification should be self-contained and include all functions which change the state of the functional unit. This especially includes both of a pair of inverse functions, which are so distinguished since they do not change the state of the functional unit when they are both performed, one after the other.
- o The function specification should be unambiguous, that is, for a definite change in state should be forseen exactly one function. With all this, "similar" functions are excluded and a clear separation of the functions is achieved.

The performance specification describes the demands on all operating means which must be fulfilled by the functional unit, especially the time and memory requirements.

Functional units play an important role in the development of complex software systems. Such systems become comprehensible and realizable if one structures and abstracts. Especially, one can and should proceed according to the principle of modularization:

- > From the modularization of the application and theoretical areas it will be attempted to derive the modularization of the systems into functional units and with this, a modularization of the data into data structures.

This principle can be made use of on many levels of a system hierarchy. An important consequence for the following work is that the modularization of multifunctional systems, with reference to a system for model description, will necessarily be reflected in a modularity of the user interface. This is forseen and desired.

From the principle of abstraction, the partial aspects of a system once considered separately may be condensed into a few. Functional units as subsystems allow abstraction from a supersystem and other subsystems. Just as much is abstracted from the structural aspects of a functional unit. So in the above-cited definition nothing is mentioned about the role of programs and computer sytems. Only the fulfillment of function and performance specification is controlling. A further abstraction concerns the storage of the internal data of a functional unit. For the function specification a general concept of the data structures for the establichment of the functions is sufficient (compare "information hiding module" [Parnas 77]).

As required by the hierarchical view, the ideas of software system and functional unit can be used equally. Following the general language usage the idea of software system shall be associated with "complex, comprehensive, global" and the idea of functional unit with "simple, comprehensible, limited". The idea "machine" as it is used in connection with VFE machine is therefore most closely equated with software system.

2.3 Use and Development Environment

The VFE machine and thus also the satellite system need to be implemented in a computer system for practical use. Therefore, two aspects of the system environment are especially important:

- the input/output (I/0) devices and
- the operating system.

For the user, the available variety of I/O devices for input and output is interesting. The input and output must be regarded as a fundamental software problem, since its unified treatment on various I/O devices is not transferable. This realization has led to a total renunciation of every language-inherent input and output concept (compare [Wirth 82]). In the model description, the input may take place by light pen, keyboard, or graphics tablet. With this in mind, it is obvious that each input medium must be supported by corresponding software measures in order to allow adequate work with the medium. While the keyboard is suited e.g. to input commands to the system, one would use a light pen in order to choose a desired command out of the possibilities displayed on the screen. The integration of so many different input media requires a concept of strict separation of syntax and semantics as will be introduced in this work (Chapter 6).

Besides the compatibility to various I/O devices,the large development costs of a software system require the largest possible independence from an operating system. For one thing, this is important with respect to a long product lifetime, during which changes in computers and operating systems should not lead to a loss of usefulness of the software. For another thing, the software system may be installed in various computers and thus become available to a wide circle of users. There must be definite minimum requirements made of an operating system in order to make possible software compatibility. These include above all the availability of a 'FORTRAN machine':

- The computer should have at its disposal a magnetic tape device in order to input the program to the machine.
- A compiler for the programming language FORTRAN [DIN 66027] {DIN = Deutsche Industrie Norm, German Industrial Standard} must be available. The choice of this programming language is based on its standardization and

widespread use. Its disadvantage as a primitive language (e.g. no support of structured programming, provision for only the simplest data types) may be balanced out by a heightened discipline in programming (compare [Schrem 74]).

- o The operating system must make available a bus which /18 can integrate independently-compiled programs into an executable module. For this is needed a virtual memory capacity, which includes the possibility for the formation of multi-user modules.
- o A loader allows the execution of the connected modules. If the corresponding I/O devices are available, then the satellite system is ready for operation.

The compatibility of the software is conceptually simplified by virtue of its I/O device-dependent and operating system-dependent parts being organized in special functional units (compare [Schrem 76, Kalb 81]). A change of the Fortran machine then affects exclusively these functional units. With these can be abstracted from the machine-dependent aspects of the satellite system.

Since the development of the satellite system is also accomplished with the help of a computer, additional requirements must be fulfilled. One can imagine very fancy systems for the development environment (compare [Ivie 77]) but here shall be mentioned only a few requirements which are indispensable, according to the experience of the author:

- o For the creation of the program a text editor is required. The input of the program over CRT (cathode-ray tube) devices available in sufficient quantity, a high-speed printer for output, a dialog-oriented operating system and a high throughput for execution of programs as well as a breakdown-free

operation over long periods of time can be regarded as a package of requirements.

- The control of the program must be possible in a data structure which is easily overviewed (compare [Ritchie 74]).
- Especially important is the constantly undisturbed access to peripherals of the computer system such as CRT devices, high-speed printers, and magnetic tape devices. A closed system is to be avoided, since it hinders the software developer from choosing his operating speed himself.

3. Fundamentals and Components of Modelling

3.1 Brief Characterzation

A model is the idealized depiction of a structure under conditions of use in such a way that the finite element method may be used directly to make calculations based on the model with the intention to interpret the results of the calculation in terms of the behavior of the structure. Thus, every application of the finite element method is characterized by the three phases of modelling, calculation, and evaluation of results. A high-performance computer with suitable software is a prerequisite for the completion of each of the three phases [Argyris 70, Schrem 71,78a].

The precursor to model development is called idealization. This consists of mentally dividing the structure into a discrete number of finite elements which suit the purpose of the calculation. This division is to be built into the modelling. For this, the mesh serves as the fundamental building block. Each mesh consists of a row of nodes which represent definite points in the visualization space. The nodes posess degrees of freedom. These stand for two dual physical quantities, of which one is unknown and the other given in each node. In structural calculations these are, depending on the component, force and displacement, or their generalized analogous quantities. In this way the boundary conditions of a structure are determined by the given forces and displacements and the unknown forces and displacements result from the calculation.

The description of each model can be divided into the following steps:

1. Description of the division of the structural model into meshes.
2. Description of the topology of the interrelationships of elements and meshes by means of nodes

3. Description of the configuration, through which the nodes are given positions and degrees of freedom of a coordinate system.
4. Description of the kinematic boundary conditions, through which the unknown displacement quantities may be separated from the given displacements.
5. Description of the elements with specific data on /20 geometry and type of engineering material.
6. Description of the loading by statement of the values of the given force and displacement quantities.
7. Calculation of the elements through creation of their discretization matrices.

Every step is chosen such that a few correlated properties of the model are coupled. The sequence results from the fact that each step in the modelling refers to the previous steps. Through this, a logicl construction of the model is achieved.

The following treatment of all steps in the description of the model is accompanied by illustrations with the commands for modelling which have arisen in the development of the system. The concept of command language is explained thoroughly in Chapter 6. Through the preferred listing of commands a direct relationship of the fundamental concepts to the practice of modelling will be produced. An example of modelling is found in Appendix A.

3.2 Element and Mesh

A fundamental concept of every model is the mesh. One differentiates between two types of meshes:

- Elementary meshes: In each of these is connected together a row of elements of any one type. Each such element is a model in miniature and is described by its

behavior with respect to physical field variables (stress and strain or force and displacement, respectively). Through simple statements about the changes in the field variables the element is idealized with a small number of degrees of freedom (up to approximately 100). Since, in a model, a large number of elements appear, elementary meshes allow an order which makes the model comprehensible in that the component parts may be regarded as a unit. Connected /21 with this is a coordinated treatment of all elements of an elementary network in the model description. The elements remain unconnected. The degree of freedom is determined from the number of elements and the respective set of degrees of freedom (compare [Schrem 78a]).

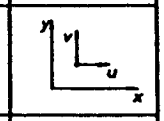
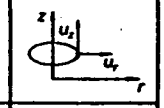
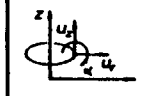
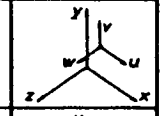
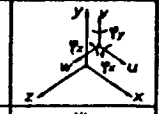
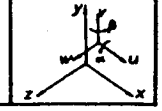
- Connecting meshes: These are created when elements of one or several elementary meshes are connected with each other. The connection results when the nodes of several elements are placed in the visualization space. This process is also called coupling. Here, the elementary meshes can be of different types. In addition, a connecting mesh is also created when one or several connecting meshes or elementary meshes are coupled together. Thus, the definition of a connecting mesh is recursive. The coupling process can be performed in as many steps as desired, until all elements in a model are coupled. Through the coupling, the degrees of freedom at the nodes of the connecting meshes are always connected directly or indirectly with the element degrees of freedom. From this results the important principle of node-connected degree of freedom:
  > In the process of coupling the meshes, only degrees of freedom connected with the nodes are counted.

The versatility of the finite element method is, above all, based on the variety of possible element types. To classify each element according to type has the advantage of being able to condense its characteristic properties into a single expression (compare Chapter 5.2). The element type includes:

- o The type name as identification.
- o The discretization in the statement function with which the progress of the field variables within the element is comprehended.
- o The organization of the element mesh.
- o The physical area of application.
- o The number of degrees of freedom and their division among the nodes.
- o The type of coordinate system in which the locations /22 of the nodes are described and according to which the degrees of freedom are oriented.

The choice of an element type is the task of idealization and an important prerequisite for modelling. This is first determined according to which physical question formulation is to be answered with a model. For this are to be considered all of the properties of an element type mentioned previously. So, for example, the order of the statement function is very important for the consideration of the expected stress changes in a model.

Configuration Space | Mesh Type | Type Name | Coordinate System | Degrees of freedom

Konfigurations-raum	Netztyp	Typ-name	Koordinaten-system	Freiheits-grade	
R^2	Ebene Kontinua	PC	x,y	u,v	
R^2	Achsensym. Kontinua	AC	r,z	u_r, u_z	
R^2	Achsensym. dicke Schalen	AT	r,z	u_r, u_z, α	
R^2	Platten	P	x,y	$w, w_{,x}, w_{,y}, w_{,xx}, w_{,yy}, w_{,xy}$	
R^3	Räumliche Kontinua	C	x,y,z	u,v,w	
R^3	Strukturen (dünne Schalen und Balken)	S	x,y,z	$u,v,w, \varphi_x, \varphi_y, \varphi_z$	
R^3	Räumliche dicke Schalen	T	x,y,z	u,v,w, α, β	

PC - planar continuum, AC- axisymmetric continuum, AT - axisymmetric thick shells, P - plates, C - spatial continuum, S - structures (thin shells and beams), T - spatial (thick shells).

Figure 3.1 Examples of mesh types for model description in structural calculations.

Because of various physical application areas, the degree of freedom and the coordinate system of various elements can be so different that they cannot be coupled by means of elementary meshes into one and the same connecting mesh because the unified description of these properties is not ensured. The element types may be so classified that this unity is achieved in each class. Each such class is designated by a mesh type, which is a property of the connecting mesh. Through this is determined (for examples see Fig. 3.1):

- o the type name as identifier.
- o the set of degrees of freedom on each node. This is the same for all nodes of a mesh.
- o the type of coordinate system, in which the location of all nodes is described and to which all degrees of freedom are related.

The elements are then coupled into a connecting mesh, so that element and mesh type allow a mutually compatible set of node variables.

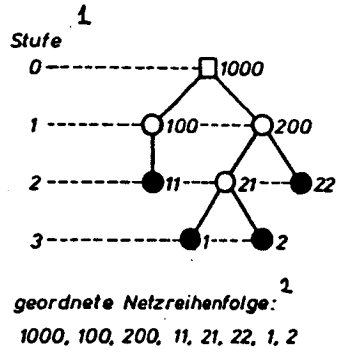
3.3 Mesh Tree

The recursive definition of the connecting mesh leads to the concept of the mesh network. Through it, the division of a model into meshes is described. Specifically, this follows the principle of unity of model and mesh tree:

> Each model is described by exactly one mesh tree and every mesh tree depicts exactly one model.

That mesh which, after the recursive coupling procedure is not itself again coupled into another connecting mesh, is called the main mesh. All other connecting meshes of a model are called partial meshes [Schrem 78a]. A mesh tree is an ordered tree with the properties (compare Fig. 3.2):

- o The roots of the tree form the main mesh. /24
- o The branches of the tree form the partial meshes.
- o The leaves of the tree form the elementary mesh.



1 - Step, 2 - ordered mesh series

Tree Part	Symbol	Mesh Type
Leaf	•	Elementary Mesh
Branch	o	Partial Mesh
Root	□	Main Mesh

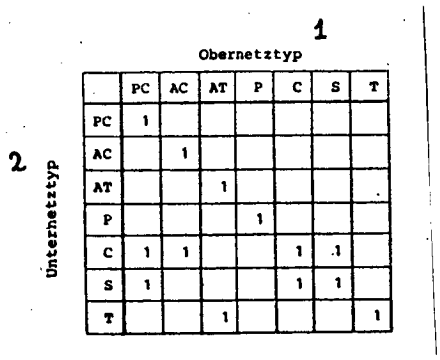
Figure 3.2 - Example of a mesh tree

The tree is called ordered, because its meshes are presented in a definite order. The order is laid down such that the meshes follow each other stepwise from right to left (as seen from the main mesh outwards). In order to designate a mesh, the mesh number will be introduced. This is a positive whole number which gives its name to the mesh. With this, each mesh is unambiguously identified within a model. The user-chosen external mesh number refers to an ordinal, internal mesh number corresponding to the mesh order.

The edges of the mesh tree give rise to a lower mesh/upper mesh relationship. Each connecting mesh is an upper mesh, the meshes coupled within it are its lower meshes. From this follows:

- Except for the main mesh, each mesh has exactly one upper mesh.
- Except for the elementary meshes, each mesh has at least one lower mesh.

The properties of the nodes of the connecting mesh are determined /25 by the mesh type. Therefore, the mesh types of two meshes which are coordinatedby a lower mesh/upper mesh relationship must be compatible (see Fig. 3.3). If the mesh type of a lower mesh represents more node variables than the upper mesh, the extra variables are called redundant node variables.

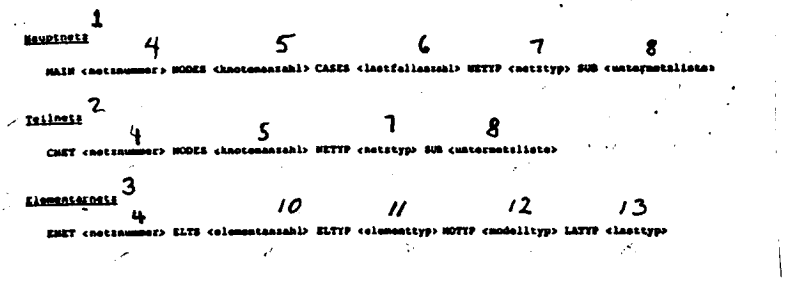


1 - Upper Mesh Type, 2 - Lower Mesh Type

Figure 3.3 - Compatibility of mesh types in the lower mesh/upper mesh relationship.

The simplest mesh tree consists of a main mesh and an elementary mesh. For the construction of complicated mesh trees many reasons can apply. One reason is the need to coordinate different types of elements and different types of elementary meshes. Another reason is purely from considerations of computer technology. That is, with idealization using very many elements, a solution is only possible by separation into several connecting meshes, since the available computer system capacity (memory space, computing time) is not sufficient for a solution in a single step. Regularities in a structure that allow its depiction using several similar meshes (e.g. with symmetry) are suited for separate idealization. Out of this results the possibility to carry out the calculation of identical discretization matrices only once. Besides this, there can also be reasons based on a more favorable model description, in which the construction of a hierarchy of connecting meshes appears advantageous. With this, one achieves the separation of structural parts which are to be changed in a later model or idealized differently (e.g. with a finer mesh or other element types. Last but not least, the aspects of

the evaluation of the results for a given construction of the mesh tree may be relevant. The description of the mesh tree occurs /26 with the commands in Fig. 3.4.



1 - main mesh, 2 - partial mesh, 3 - elementary mesh, 4 - mesh number, 5 - number of nodes, 6 - number of loadings, 7 - mesh type, 8 - lower mesh lists, 10 - number of elements, 11 - element type, 12 - model type, 13 - load type.

Figure 3.4: Commands for mesh tree description. (For examples, see Figure 4.5).

3.4 - Nodes and Incidences

In order to coordinate node variables with a node, the node must be identifiable. This purpose is served by node numbers. These are arbitrary positive whole numbers which give each node a name. Within a connecting mesh this node name must be unambiguous, but the same number/name may be used to identify nodes in different meshes. The order of the node variables at a node is determined by the mesh type. The order of node variables in a connecting mesh results from the definition of a nodal order. For this, the n node numbers of a

mesh are depicted by the ordinal numbers from 1 through n. In order to differentiate them, the former will be called external node numbers; the latter, internal node numbers. The arrangement of the node variables in the connecting mesh results in the so-called canonical order of node variables.

The coupling of elements and meshes is accomplished with incidences. One differentiates:

- o node incidences and
- o incidences of degrees of freedom.

The node incidences facilitate the coupling in visualization /27
space, the incidences of degrees of freedom facilitate the coupling in the dual space of the degrees of freedom. Two nodes are incident if they occupy the same point in the visualization space. The coupling of upper and lower meshes is then simply described by the listing of the ordered pairs of all incident lower and upper nodes. Each node of a lower mesh is incident with either zero or exactly one upper mesh node. With respect to multiple incidences, the principle of unambiguous coordination of node and point applies:

- > Different lower mesh nodes may not be incident with the same upper mesh node if they do not represent the same point.

Each edge of the mesh tree represents the availability of node incidences. Specifically, the following apply:

- o All elementary mesh nodes are incident with upper mesh nodes.
- o In each partial mesh exist incident nodes; otherwise, the mesh tree would fall apart into disjunct graphs. This would violate the principle of unity of model and mesh tree.
- o No main mesh node can be incident, because there is no upper mesh.

Since degrees of freedom are bound onto nodes, incidences of degrees of freedom are only possible on incident nodes.

3.5 Topology

3.5.1 Description

In the following, the coupling of the elements in a connecting mesh will be examined. Topology of such a net means the totality of all aspects of graph theory of this part of the model. All aspects of the degree of freedom or the location and size of the participating elements is disregarded.

/28

The topological properties of elements are determined by element type and include:

- o the dimensions of the elements; that is, whether node, edge, planar, or solid elements are concerned.
- o the number and order of the element nodes.
- o the number of edges and surfaces and their description as polygonal inscription through the nodes concerned.

Nodes, edges, surfaces, and solids are the topological units of a mesh.

The coupling of the elements is described by element incidences. In this,all node incidences of an element with its upper connecting mesh are included. For description, it suffices to count up the node numbers of all connecting meshes with one incident element, in the element-dependent order. Corresponding to the nodes, an element number is used to identify each element. The determination of an element order leads also to a differentiation between internal and external element numbers.

How many elements are associated at a connecting mesh node is denoted by a degree of node incidence. In summary, the topology of a given connecting mesh is described by:

- o the quantity and the numbers of the connecting mesh nodes,
- o The quantity and type of the applied elements, and
- o the node incidences for each element.

These descriptive quantities make possible the construction of a very general class of mesh which, based on its physical background, has some limitations. So, e.g. interpenetration of elements is not possible and inhomogeneities may only appear in the mesh when the underlying structure is homogeneous. A mesh will thus be called topologically correct if all such limitations are fulfilled. However, for a given structure there can be very many topologically correct meshes. Which one will be chosen in a given case depends largely upon physical considerations. These can be influenced by the applied loads, by symmetry of structure or loading or the desire for a simpler description.

/29

3.5.2 Degree of Incidence

In order to judge the topological correctness of a mesh, the idea of the node incidence degree will be generalized to the other topological units. Thus, an edge incidence degree is defined which denotes how many elements two nodes which determine an edge have in common. The same applies for surface and solid incidence degrees. Besides this, it has proven to be practical, in the establishment of the topology of an elementary mesh, to consider middle nodes separately and to connect no edges with such nodes. One differentiates solid, surface, and middle edge nodes. By this, all common elements may be related back to only six different fundamental topological elements:

- o one edge element,
- o two surface elements (triangular, quadrilateral)
- o three body elements (tetrahedron, pentahedron, hexahedron).

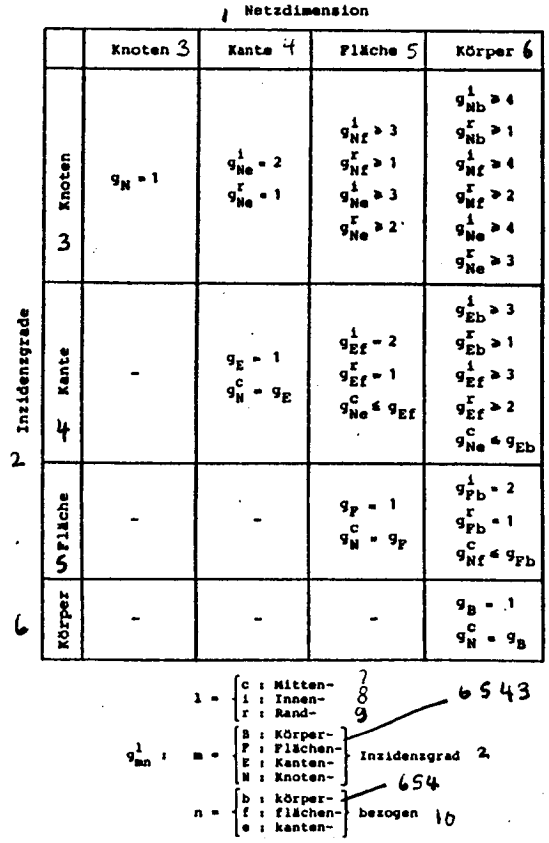
With the help of the incidence degree the following rules for topologically-correct meshes may be formulated:

- o The degree of incidence of (edge, surface, and solid) elements is 1. Out of this results the unambiguity of element incidences.
- o The degree of incidence of a surface is either 1 (so-called outer surfaces) or 2 (so-called inner surfaces or else surface elements lying in between solid elements.
- o With two-dimensional meshes the degree of incidence of edges is either 1 (so-called outer edges) or 2 (so-called inner edges or edge elements lying in between surface elements.
- o If two solids have in common three nodes which are not middle nodes, then they also have a surface in common which includes the three nodes, otherwise a crack must exist between the two solid elements and hence in the structure itself.
- o If two surfaces have in common two nodes which are not middle nodes, then they also share a common edge between these nodes.
- o Solid middle nodes have a degree of incidence of 1.
- o Either the surface and edge nodes have the incidence degree of the surface to which they belong, or else a hole will appear between elements and also on the structure itself.

/30

One obtains further rules about topology if one ascertains the degree of incidence not only with respect to elements, but also considers

degrees of incidence related to solids, surfaces, and edges. A summary of all rules which can be made about degree of incidence is shown in Fig. 3.5. For this, it is generally assumed that elementary meshes of different types are coupled.



1 - mesh dimension, 2 - degree of incidence, 3 - node, 4 - edge, 5 - surface, 6 - solid, 7 - middle, 8 - inner, 9 - edge, 10 - related .

Figure 3.5: Degrees of incidence for topological units in finite element meshes

3.5.3 Euler-Poincaré Characteristic

While the degree of incidence allows one to make statements about every topological unit -- that is, a microscopic view of the mesh, a global view of the mesh may be performed with the help of the numbers of topological units. According to the formula for three-dimensional networks from Euler and Poincaré [Schubert 69], the following applies:

$$n - e + f - b = C \tag{3.1}$$

where n, e, f, and b are the numbers of nodes, edges, surfaces, and solids, repectively, in a mesh. Thus C is the Euler-Poincaré characteristic of the mesh to which the formula is applied. If one ascertains the characteristic for the fundamental topological elements, this always results in a value of 1 (see Fig. 3.6). As long as such elements are simply arranged together in a connecting mesh, the characteristic does not change, since it is only dependent on the topological continuity of the mesh.

The nominal value of the characteristic results simply from the form of the structure which is idealized by the mesh. This nominal value is called the topological condition number and is independent from each idealization of a property of the structure. For this applies

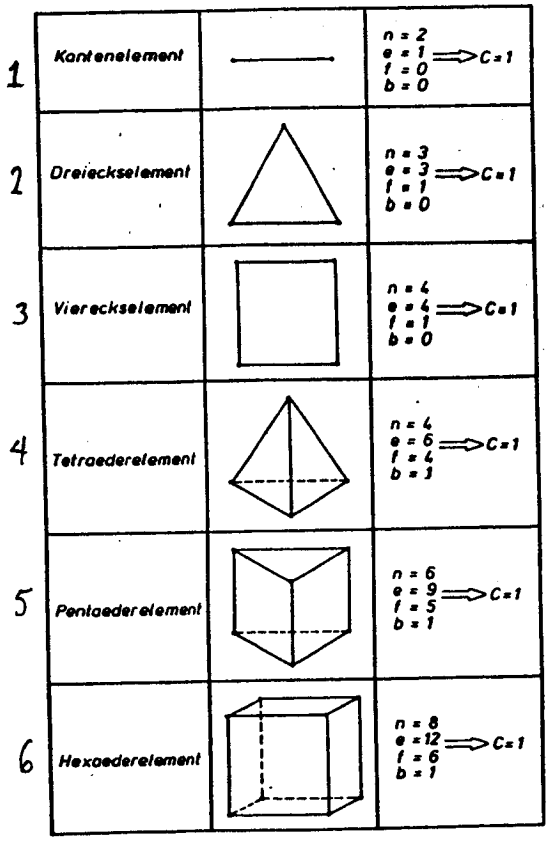
$$C_K = k - r - h \tag{3.2}$$

where k is the number of components of the mesh which among themselves exhibit no coupling. k is commonly 1. The number of cracks and holes in the structure is given by r. Cracks include every disturbance of the topological relationship in the sense that a closed curve which

cannot be condensed to a point may be constructed on the outside of the structure. The number of hollow spaces in three-dimensional models is given by the number h. Examples of numbers for topological conditions (for k=1) are the following:

- o circle, cylinder, Möbius strip (r=1, h=0): $C_K=0$
- o hollow torus (r=2, h=1): $C_K=0$
- o solid sphere (r=0, h=0): $C_K=1$
- o hollow sphere (r=0, h=0): $C_K=2$
- o two-cell fuselage shape with window cut-out (r=3, h=0): $C_K=-2$

/32



1 - edge element, 2 - triangular element, 3 - quadrilateral element, 4 - tetrahedral element, 5 - pentahedral element, 6 - hexahedral element.

Figure 3.6: Characteristics of elements (n, e, f, b, : numbers of nodes, edges, surfaces, and solid bodies, respectively)

In summary the necessary condition for topological correctness of an idealization is

$$C = C_K \tag{3.3}$$

Every deviation from this requirement indicates a defect in the topology of the mesh. A common reason for such defects are incorrect element incidences. Either an element is identified with the wrong connecting mesh node numbers or the order of these numbers is incorrect.

/33

If not all nodes of the topologically-correct mesh are used in the element incidences, n is reduced and, in general, from Eq. (3.1), $C < C_K$. It should be now assumed this error does not occur, e.g. n is constant. Incorrect element incidences in meshes of surface elements then lead to an increase in the number of edges since n = f = constant and $-e = C - n - f$, and again $C < C_K$. With solid elements, however, the characteristic remains unchanged, since the same number of edges as surfaces intersect on every element node, so n = b = constant and $-e + f = C - n - b$, $C = C_K$ applies. However, the outer surface is enlarged by errors in the element incidences. The outer surface is composed of all outer surfaces of a mesh. For its Euler-Poincaré characteristic

$$C_R = n_R - e_R + f_R \tag{3.4}$$

/34

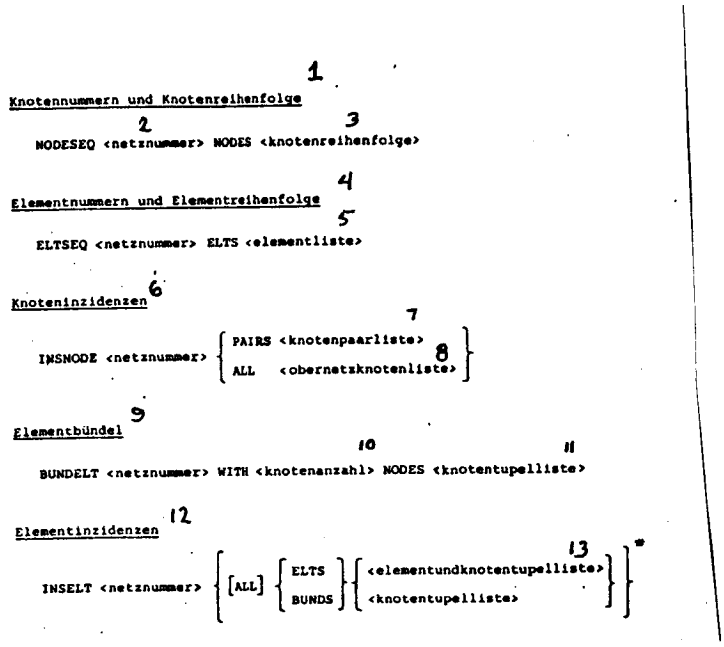
It must therefore also be true (as in Eq. (3.3)) that

$$C_R = C_{KR} \tag{3.5}$$

with

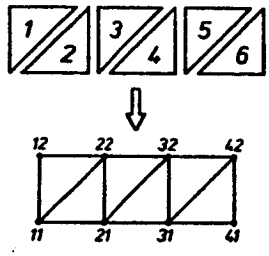
$$C_{KR} = k_R - r_R + h_R \, . \tag{3.6}$$

The index R designates here the outer surface of the mesh.



1 - node numbers and node sequence, 2 - mesh number, 3 - mesh sequence, 4 - element numbers and element sequence, 5 - element list, 6 - node incidences, 7 - node pair list, 8 - upper mesh node list, 9 - element bundle, 10 - number of nodes, 11 - node type list, 12 - element incidences, 13 - element and node type list

Figure 3.7: Commands for the description of the topology



1 Elementarnetz 10

2 Verbundnetz 1000

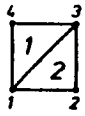
3 Knotennummern und Knotenreihenfolge

```
NODESEQ 1000 NODES 11,12,21,22,31,32,41,42
```

4 Elementnummern und Elementreihenfolge

```
ELTSEQ 10 ELTS 1,2,3,4,5,6
```

5 Beschreibung der Elementinzidenzen mit Hilfe eines Elementbündels



6 Elementbündel

```
BUNDELT 10 WITH 4 NODES 1,3,4 / 3,1,2
INSELT 10 ALL BUNDS 11,21,22,12 / 21,31,32,22 / 31,41,42,32
```

7 Diese Beschreibung der Elementinzidenzen ist gleichwertig mit

```
INSELT 10 ALL ELTS 11,22,12 / 22,11,21 / 21,32,22 /
                32,21,31 / 31,42,32 / 42,31,41
```

1 - elementary mesh, 2 - connecting mesh, 3 - node numbers and node order, 4 - element numbers and element order, 5 - description of the element incidences with the help of an element bundle, 6 - element bundle, 7 - this description of the element incidences is equivalent to

Figure 3.8: Simple example for the description of the topology.

In summary, it may be said that a mesh may be called topologically correct if the rules about the degree of incidence and the conditions (3.3) and (3.5) are fulfilled. These are however only prerequisites for the determination as to whether an idealization is correct. For more thorough examination the node coordinates are needed. Then the user can determine by inspection the correctness of the idealization, e.g. with the help of a graphical depiction of the mesh.

The description of the topology is accomplished with the commands shown in Fig. 3.7. Attention is particularly called to the possibilities of description of element incidences using element bundles. With these a repeating pattern of coupled elements out of an elementary mesh may be assembled, e.g. two triangles to every quadrilateral. Through the naming of external upper mesh node numbers for the nodes of a bundle, the bundle may then be placed several times into the upper mesh. Depending on the model, this can be connected with a considerable simplification of the description. A simple example of the descriptive possibilities is shown in Figure 3.8.

3.6 Configuration

3.6.1 Rules for Description

The configuration includes the two properties of nodes:

1. Node coordinates: location of nodes.
2. Node basis: reference directions for the degree of freedom at the nodes.

While topology describes a mesh that could still manifest itself arbitrarily in the visualization space, the configuration describes exactly one manifestation. It must therefore be geometrically compatible, since incorrect coordinates could cause contradictions to

the shape of the model or its elements to arise, e.g. a violation of the unambiguous coordination of node and point, the placement of all nodes of a solid element in a single plane, or the creation of cracks. /36

The configuration has a large influence on the quality of an idealization which is described by the so-called discretization error. Ths denotes all deviations of the calculated behavior of the structure from the actual behavior, which can be traced back to the idealization. In order to keep discretization error as small as possible, some regulations for the construction of meshes can be heeded (compare [Taig 75, Melosh 79]), e.g.:

- o Symmetries and regularities of the structure should be reflected in the mesh. (Requirement of structural equivalence of model and structure).
- o Each element should have edges of as equal length as possible, and the edge lengths of an element should differ little from neighboring elements. (Requirement of formal equivalence).
- o With respect to the boundary conditions, the elements should be chosen smaller where large stress gradients occur (Requirement of proportionality of effort).

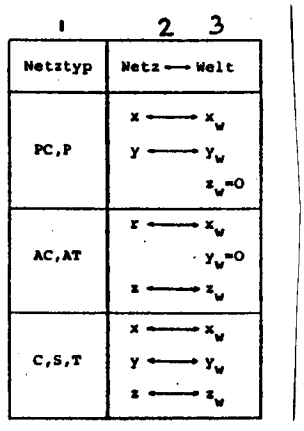
Since the user can, in reality, describe very many different models of the same structure, such regulations simplify the choices.

3.6.2 Node Coordinates

The mesh type determines the corresponding coordinate system for each connecting mesh (compare Fig. 3.1). This mesh coordinate system determines type and canonical series of the coordinate axes. Node coordinates referring to the mesh coordinate system are also called mesh coordinates. For all meshes of a model exists exactly on

global reference system: the world coordinate system. It is always three-dimensional and Cartesian. Node coordinates are designated world coordinates with respect to the world coordinate system. For the conversion from mesh coordinates of a given system into world coordinates and vice versa, a set of functions is available (see Fig. 3.9). Only mesh coordinates which allow such a reversible unambiguous coordination are provided for.

/37



1 - mesh type, 2 - Mesh, 3 - world
Figure 3.9: Coordination of mesh and world coordinates

The location of a mesh coordinate system is described by a mesh basis. Each mesh is allotted exactly one mesh basis. Through this, the mesh coordinates may be transformed, and meshes with the same configuration may easily be placed at different locations with in a model. The model sizes of mesh basis B_N and mesh coordinates x_N are formed so that for world coordinates x_W it applies that:

$$x_W = B_N x_N \qquad (3.7)$$

Through the introduction of Cartesian world coordinates, it becomes possible to compare the node coordinates of meshes of different types. With this, a unified foundation for the creation of graphic model depictions is given.

In the description of certain parts of the mesh it is often advantageous to express the coordinates in terms of a local coordinate system which does not agree with the mesh coordinate system (e.g. cylindrical, spherical, and polar coordinate systems). Since for the local coordinates so described it is not necessary to have a reversible transformation to the mesh coordinate system, a variety of other coordinate systems may find use (e.g. toroidal, bicylindrical, or elliptical coordinate systems). For these only a functionto convert to mesh coordinates is needed. The location of the local coordinate system compared to the mesh coordinate system is described by a local basis. Local coordinate systems and local bases serve /38 to simplify creation of coordinates and have no other further meaning other than for the description of the configuration.

Besides these, for the scaling of coordinates, there is also available the means of description of the units of measure. With this, calculations may be carried out with various measuring systems and size relationships. The used can choose from a variety of measuring units (e.g. millimeter, inch, degree, etc.). All length units are converted to meters and all angles are converted to radians (according to [DIN 1301]).

In summary, there exists a relation which is summarized by the following conversion scheme (compare Fig. 3.10):

U1: Conversion of measuring units.
U2: Conversion of the local to the mesh coordinate system.
U3: Transformation of the local coordinate system to the mesh coordinates with the help of the local basis.

U4: Conversion of the mesh coordinate system to the world coordinates with the help of the mesh basis.

The descriptive quantities relevant for the configuration are:

- o the mesh coordinates (created by conversions U1 - U3) and
- o the mesh basis.

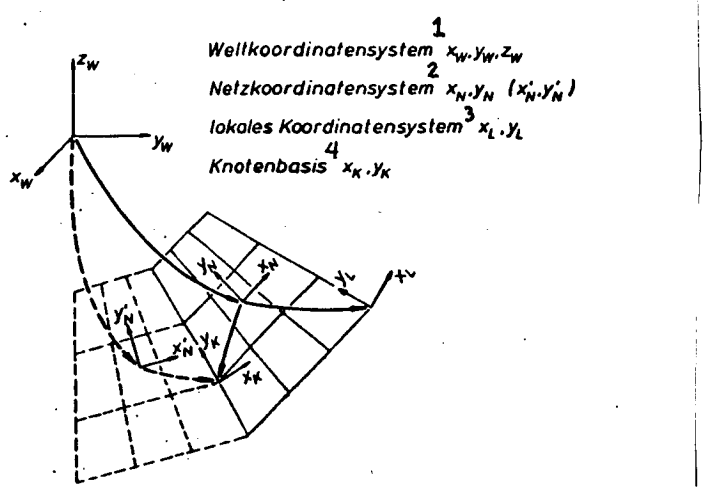
The mesh coordinates must now not be declared in all meshes of a model; rather, from the principle of complete coordinate description it may be proceeded:

> All nodes are to be provided with coordinates. However, for incident nodes, coordinates need only be described once.

The choice of the meshes for the description is up to the user. It has, however, proven to be advantageous to describe the coordinates of all nodes of the upper meshes of the elementary meshes. If incident nodes of different upper meshes do not receive the same coordinates, that is an indication of an incorrect description. In this /39 connection, the difference between defined node coordinates and undefined node coordinates is important. Defined node coordinates are all coordinates supplied by the user, all others are undefined.

The description of a mesh basis is only required if a difference exists in the locations of the mesh and the world coordinates. Then the mesh basis is described with respect to the world coordinate system by means of translations and rotations. The same is true of a local basis with respect to the mesh coordinate system. The mesh coordinates and mesh bases described suffice to give the world coordinates of all nodes automatically. Thus,the principle of complete coordinate description comes to use in the following deduction:

> Incident nodes have identical world coordinates.



1 - world coordinate system, 2 - mesh coordinate system, 3 - local coordinate system, 4 - node basis

Figure 3.10: Coordinate systems and bases for the description of the configuration.

/40

The world coordinates are determined by the following procedure. This will be called consolidation of coordinates:

1. Computation of the world coordinates from the defined mesh coordinates (created by the conversions U4 and U5).
2. Transferring the world coordinates upwards:
   In a traversing of the mesh tree from the elementary mesh to the main mesh (according to the ordered mesh sequence) all defined world coordinates are tranferred into the next higher upper mesh in each step via the node incidences. The agreement of redundant coordinates with all nodes of a mesh is demanded (compare Chapter 3.3). Deviations from defined upper mesh coordinates can be an indication of incorrect node incidences or an incorrect description of coordinates.

3. Transferring the world coordinates downwards:
   By the reverse traversing from the main mesh to the connecting meshes of the highest rank, all defined world coordinates are transferred into the next-lowest mesh via the node incidences. Agreement is again demanded for redundant coordinates. Deviations from defined lower mesh coordinates can again indicate errors in description.
4. World coordinates for elements: via the element incidences all element nodes receive the world coordinates from the corresponding upper mesh.
5. World coordinates for middle nodes: The element meshes of some types provide for middle nodes. For their world coordinates the following rules apply:
   - o If they are undefined, they are determined from the corner nodes by interpolation.
   - o If they are defined, no change occurs. If they correspond to the interpolated coordinates, surfaces and edges will be flat or straight, respectively. Otherwise, they will be bent.
6. If world coordinates are determined for middle nodes, steps (2) and (3) are repeated.

/41

After consolidation undefined world coordinates may not exist for any nodes of any mesh. From the compatibility of mesh and element type (Chapter 3.2) it follows that an element coordinate system is always identical with the mesh coordinate system of the corresponding upper mesh. Therefore, the consolidation still must be completed by:

7. Conversion of the world coordinates for the element nodes in the element coordinate system.

3.6.3 Regular Node Distribution

Commonly, for reasons of simplicity, a model is idealized so that several nodes of a mesh are regularly distributed in the visualization space. The term "regularity" is used because the measurement numbers of the coordinates can be given by an arithmetic or geometric series in a definite reference system. Through suitable coordinate systems it is possible to describe the location of nodes by means of characteristic geometric loci, e.g. with:

- straight lines in Cartesian coordinate systems
- circles and straight lines in cylindrical coordinate systems.
- spherical surfaces and straight lines in spherical coordinate systems.

Moreover, a reference system is given by interpolation functions, with the help of which the coordinates may be indirectly written in parameter form. According to the number of necessary parameters one differentiates:

- line parameter ξ : With this the nodes lie on an arbitrarily curved line.
- surface parameters ξ, η : The nodes lie on an arbitrarily curved surface which forms a topological triangle or quadrilateral.
- space parameters ξ, η, ζ : The nodes are distributed in space. Tetrahedra, pentahedra, and hexahedra are used as topological regions.

The interpolation functions facilitate the desciption of a regular distribution of the nodes in parameter space in the framework of a certain coordinate system. The form of the interpolation function depends on

- o the choice of functions (e.g. Lagrangian polynomials).
- o the topological region and hence the number of parameters.
- o the number and arrangement of so-called support nodes whose coordinates must exist in explicit form and not in parameter form.

The more support nodes available, the higher the order of the function and the more exact is the approximation of a geometric locus. In reference to the arrangement of support nodes, one differentiates two cases:

- o The support nodes are distributed over the entire topological region.
- o The support nodes lie only on the edge of the topological region. Thus one is saved having to specify support nodes in the interior.

The location of the support nodes can control the distribution of support nodes in the region and thus allow the creation of patterns of refinement. Obviously, there exists a direct relationship between interpolation functions for certain topological areas and the statement functions for elements. They are even identical, if the topology is the same, if Lagrangian polynomials are used with the elements.

Let x_i be the coordinates of the support node i, $x = \{x_1, x_2, \ldots x_m\}$ the coordinates of all m support functions, f the set of all m interpolation functions and ξ_k the parameters of a node k in the given topological region. Thus f must fulfill the following conditions (compare Appendix C):

1. $f^i(\xi_i)=1$ and $f^i(\xi_j)=0$ for $1 \leq i, j \_ m$ and $i \neq j$.
2. $\sum_{i=1}^{m} = 1$.

Then for the coordinates of this node

$$x_k = f^T(\xi_k)\, x \qquad (3.8$$

For the topological regions the results in detail are:

1. For the line with (n+1) support nodes result functions of the nth order for each support node i (Lagrangian polynomials, compare [Rutishauser 76]):

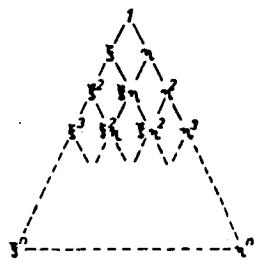
$$f_n^i(\xi) = \prod_{\substack{j=1 \\ j\neq i}}^{n+1} \left(\frac{\xi - \xi_j}{\xi_i - \xi_j}\right) \qquad (3.9)$$

with $0 \leq \xi \leq 1$.

2. For a topological triangle with support nodes in the entire area after [Argyris 68] applies:

$$\Delta f_n^{i,j}(\xi,\eta) = f_{i-1}^i(\xi) \cdot f_{j-1}^j(\eta) \cdot f_{k-1}^k(1-\xi-\eta) \qquad (3.10)$$

with i+j+k = n+3. Here i and j are the numbers of the support nodes in the ξ-direction and the η-direction respectively. The terms of the function may be depicted in visual form as a Pascal's triangle:



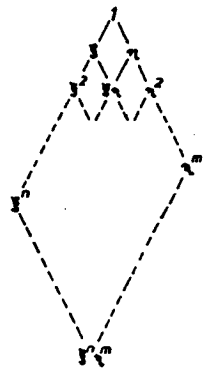
3. For a topological quadrilateral there exist for support nodes in the entire area Lagrangian polynomials according to equation 3.9:

$$\square f_n^{i,j}(\xi,\eta) = f_n^i(\xi)\, f_n^j(\eta) \quad . \tag{3.11}$$

If n is the order of the function in the ξ-direction and m the order in the η-direction:

$$\square f_{n,m}^{i,j}(\xi,\eta) = f_n^i(\xi)\, f_m^j(\eta) \tag{3.12}$$

The terms of this function can be depicted in the following manner using Pascal's triangle:



If only edge nodes are used as support nodes, it results that (compare [Coons 67]):

$$\square f_{n,m}^{i,j}(\xi,\eta) = f_n^i(\xi) f_1^J(\eta) + f_m^j(\eta) f_1^I(\xi) - f_1^I(\xi) f_1^J(\eta) \tag{3.13}$$

with

$$I = \begin{cases} 0 \; ; \; 1<i<(n+1) \\ 1 \; ; \; i=1 \\ 2 \; ; \; i=(n+1) \end{cases} \quad ; \quad J = \begin{cases} 0 \; ; \; 1<j<(m+1) \\ 1 \; ; \; j=1 \\ 2 \; ; \; j=(m+1) \end{cases} \tag{3.14a}$$

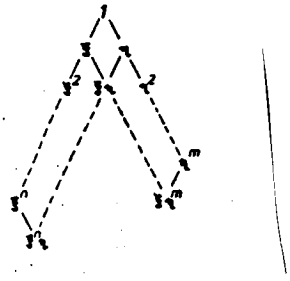
or

$$\begin{aligned} I &= \max\left[(\mathrm{div}(i-1,n)-\mathrm{mod}(i-1,n)+1),0\right] \\ J &= \max\left[(\mathrm{div}(j-1,m)-\mathrm{mod}(j-1,m)+1),0\right] . \end{aligned} \tag{3.14b}$$

respectively.

/45

For the depiction in Pascal's triangle results:



Through the exclusive use of edge nodes one obtains no coupling of the functions in the ξ- and η-directions. Thus, the order of the functions $f_n^i(\xi)$ can be changed from $j=1$ with $n=n_1$ to $j=m+1$ with $n=n_2$. This applies analogously for $f_m^j(\eta)$. This yields:

$$\square f^{i,j}_{(n,m)_{1,2}}(\xi,\eta)=f^i_{n_J}(\xi)f^J_1(\eta)+f^j_{m_I}(\eta)f^I_1(\xi)-f^I_1(\xi)f^J_1(\eta) \tag{3.15}$$

with I,J accordingly from Eq. (3.14). (For proof, see Appendix C).

4. In a tetrahedral region, analogous to the triangle, applies:

$$\bullet f_n^{i,j,k}(\xi,\eta,\zeta)=f_{i-1}^{i}(\xi)f_{j-1}^{j}(\eta)f_{k-1}^{k}(\zeta)f_{l-1}^{l}(1-\xi-\eta-\zeta) \tag{3.16}$$

with i+j+k+l = n+4. It has been shown that, in triangular and tetrahedral regions, for orders greater than 2, no set of interpolation functions may be given in which the support nodes lie only on the edge because the three or four function terms from Eq. (3.10) and Eq. (3.16), respectively, are not linearly independent.

5. In a pentahedral region,the functions are created by a combination of equations (3.10) and (3.11):

$$\blacktriangledown f_{n,m}^{i,j,k}(\xi,\eta,\zeta)=\Delta f_n^{i,j}(\xi,\eta)f_m^{k}(\zeta) \tag{3.17}$$

/46

or a combination of equations (3.10) and 3.15):

$$\blacksquare f_{n_1,n_2,m}^{i,j,k}(\xi,\eta,\zeta)=\Delta f_{n_{\overline{k}}}^{i,j}(\xi,\eta)f_1^{\overline{k}}(\zeta)+f_m^{k}(\zeta)\Delta f_1^{\overline{\imath},\overline{\jmath}}(\xi,\eta) -\Delta f_1^{\overline{\imath},\overline{\jmath}}(\xi,\eta)f_1^{\overline{k}}(\zeta) \tag{3.18}$$

with $\overline{\imath},\overline{\jmath},\overline{k}$ from Eq. (3.14).

6. In a hexahedral region in the case of support nodes also in the interior applies, analogous to Eq. (3.11):

$$\blacksquare f_{n,m,l}^{i,j,k}(\xi,\eta,\zeta)=f_n^{i}(\xi)f_m^{j}(\eta)f_l^{k}(\zeta) \quad . \tag{3.19}$$

Considering only nodes on the edges on the hexahedron and assigning a different order to the polynomial on each edge yields, analogous to Eq. (3.15):

$$
\begin{aligned}
&\mathfrak{P}\, f^{i,j,k}_{((n,m,l)_{1,2})_{1,2}}(\xi,\eta,\zeta) = \\
&f^{i}_{n_{K,K}}(\xi) f^{j}_{m_{K,K}}(\eta) f^{K}_{1}(\zeta) + f^{i}_{n_{J,J}}(\xi) f^{J}_{1}(\eta) f^{k}_{1_{J,J}}(\zeta) \\
&+ f^{I}_{1}(\xi) f^{j}_{m_{I,I}}(\eta) f^{k}_{1_{I,I}}(\zeta) - f^{i}_{n_{J,K}}(\xi) f^{J}_{1}(\eta) f^{K}_{1}(\zeta) \\
&- f^{I}_{1}(\xi) f^{j}_{m_{I,K}}(\eta) f^{K}_{1}(\zeta) - f^{I}_{1}(\xi) f^{J}_{1}(\eta) f^{k}_{1_{I,J}}(\zeta) + f^{I}_{1}(\xi) f^{J}_{1}(\eta) f^{K}_{1}(\zeta)
\end{aligned}
\qquad (3.20)
$$

with I,J,K according to Eq. (3.14).

Thus, Lagrangian interpolation functions are known for all six different fundamental topological areas. The parametric form of the coordinate description represents a powerful broadening of the palette of transformations of coordinates. It is noted, however, that the oscillations of the function values between the support nodes in the interpolation of node coordinates, commonly feared in Lagrangian interpolations, have no significance. For one thing, node coordinates always have discrete values, and for another thing, a sufficient quantity of support nodes may always be named which exactly determine the expected locus of the interpolated nodes (for an example, see Appendix A).

/47

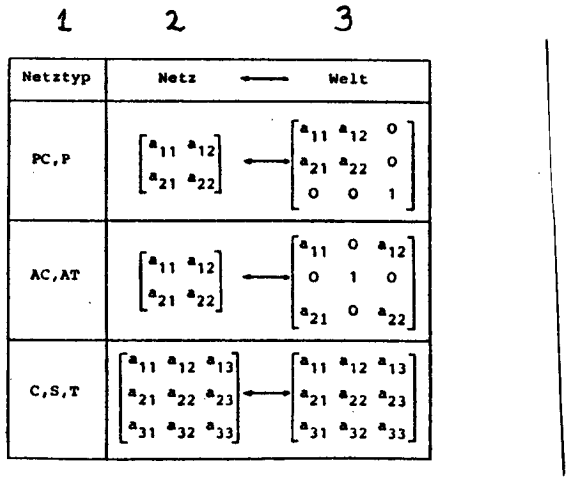
3.6.4 Node Bases

For the orientation of the degrees of freedom on a node the respective mesh basis presents itself first of all. The unified treatment is in many cases insufficient because for one thing, kinematic boundary conditions can exist which are not parallel to the axes of the mesh basis. For another thing, different meshes can have various bases, through which can arise incompatibilities on incident

nodes. For the treatment of directions of degrees of freedom the concept of node bases is available. Each node is a carrier of such a basis. It represents a pure rotation, because the translation component is included by the node coordinates. Analogous to node coordinates, one differentiates mesh node bases and world node bases, which can be transformed through the mesh basis and over the mesh coordinate system (see Fig. 3.11).

In the description of the mesh node bases, it applies that unspecified node bases determine the same directions as the mesh basis. Moreover, the same principles apply as for the node coordinates:

> For incident nodes, bases need only be described once.



1 - mesh type, 2 - mesh, 3 - world

Figure 3.11: Association of mesh node bases and world node bases.

and

> incident nodes have identical world node bases.

From this results the following procedure for the consolidation of node bases:

1. Calculation of the world node bases from the mesh node bases.
2. Upwards transferral of the world node bases:
   In a traverse of the mesh tree from the elementary mesh to the main mesh, all world node bases are transferred via the node incidences into the respective upper mesh. Deviations from specified upper mesh node patterns can be an indication of incorrect node incidences or an incorrect description of the node bases.
3. Downwards transferral of the world node bases:
   In a traverse of the mesh tree in the other direction from the main mesh to the connecting meshes of the highest rank, the node incidences of all world node bases are transferred into the respective lower mesh. Again, deviations from specified lower mesh node bases can indicate errors in description.
4. World node bases for elements: With the help of element incidences all element nodes receive the world node bases from the accompanying upper mesh.
5. Conversion of the node bases of the elements from world coordinate systems to the element coordinate system.

The description of the node bases is made simple by using the known node coordinates and the local basis. The following possibilities are considered sufficient:

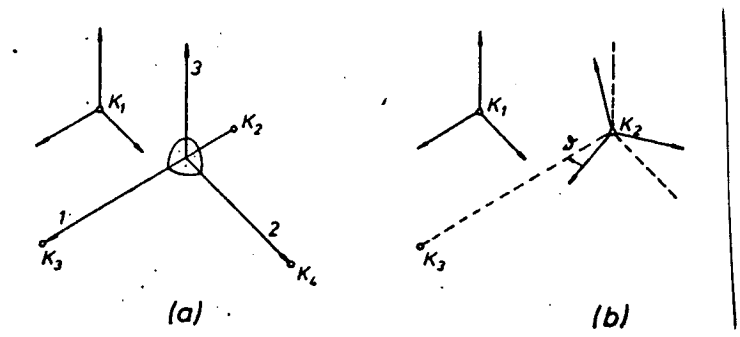


Figure 3.12: Defining process for node bases.

I. A node basis is in agreement with the directions of the local basis.

II. In the three-dimensional case, a basis on node K_1 is defined by three additional nodes K_2, K_3, and K_4 (Fig. 3.12a). The first axis of the node basis points from K_2 to K_3; the second axis is perpendicular to the first in the direction of K_4; the third axis creates with the others a system of directions. The establishment of the node bases happens with the Schmidt orthogonalization process (compare [Kowalsky 79]. Notable special cases are: /49

a) The second axis of the node basis points from K_2 to K_3; the other axes are revealed by cyclic substitution.
b) $K_3 \equiv K_1$
c) The location of K2 is the location of the local basis.

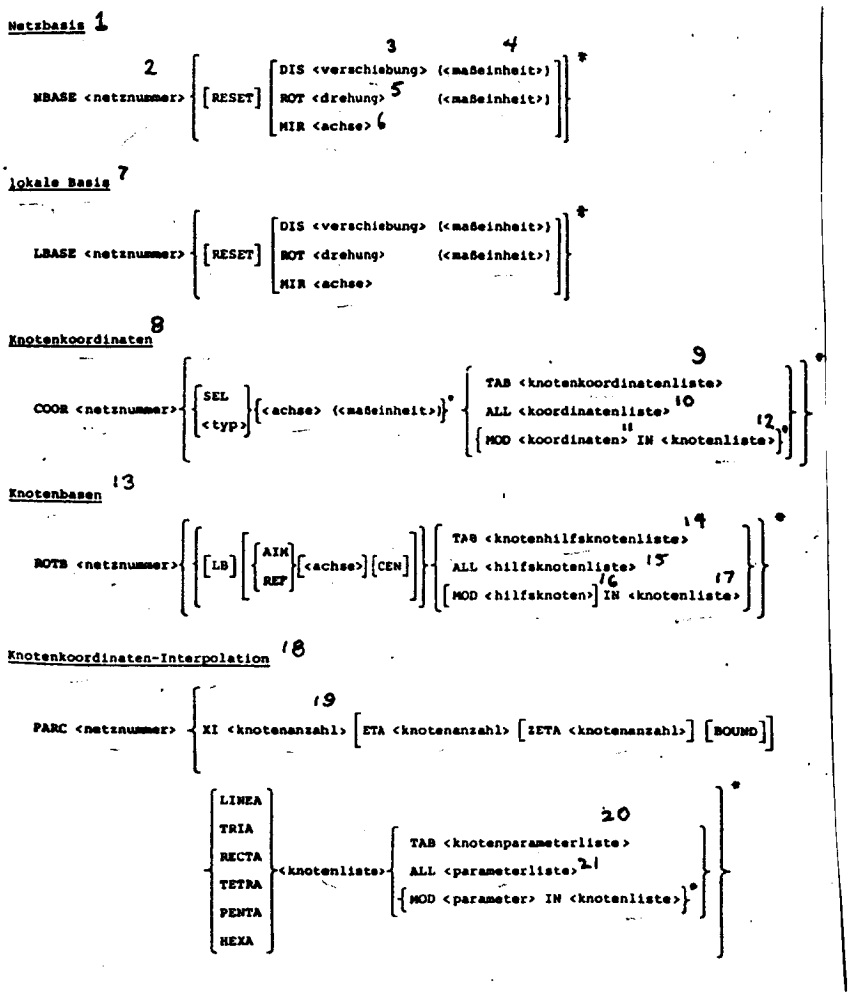
According to this process, in the two-dimensional case two nodes suffice for the description of the node basis.

III. A basis on node K_1 is, in the three-dimensional case as in the two-dimensional case, determined by two further nodes K_2 and K_3 (Fig. 3.12 b). Here, the basis of node K_2 is aligned with a definite axis in the direction toward node K3. The establishment of the node basis occurs according

to a process given by [Argyris 82]. Special cases are:

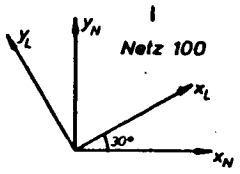
a) $K_3 \equiv K_1$.
b) According to locations and directions, K_2 is the local basis.

Interpolations of node bases are not provided for, since this generally results in a loss of the orthonormality of the bases. In summary, all commands for description of the configuration are summarized in Figure 3.13. The description possibilities given there are best shown by means of an example, as in Appendix A. For explanation of the separate commands, further examples are shown in Figure 3.14.



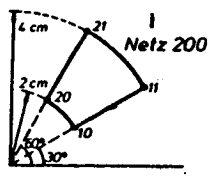
1 - mesh basis, 2 - mesh number, 3 - displacement, 4 - unit of measure, 5 - rotation, 6 - axis, 7 - local basis, 8 - node coordinates, 9 - node coordinate list, 10 - coordinate list, 11 - coordinates, 12 - node list, 13 - node basis, 14 - node auxiliary node list, 15 - auxiliary node list, 16 - auxiliary nodes, 17 - node list, 18 - interpolation of node coordinates, 19 - number of nodes, 20 - node parameter list, 21 - parameter list

Figure 3.13: Commands for the description of a configuration



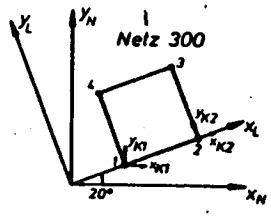
Rotation of the local coordinate system with respect to the mesh coordinate system:

```
LBASE 100 ROT 30(GRAD)        2
```



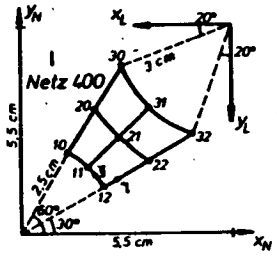
Description of the node coordinates in polar coordinates:

```
COOR 200 POLAR R(CM) PHI(GRAD)    2
     TAB 10 2,30/11 4,30/
         20 2,60/21 4,60
```



The following descriptions of the node basis on node 2 are equivalent:

```
I:   LBASE 300 ROT 20(GRAD)    2
     ROTB 300 LB IN 2
II:  ROTB 300 AIM Y TAB 2 2,3
III: ROTB 300 REF X TAB 2 1,2
```



Description of the location of the inner edge:

```
COOR 400 POLAR R(CM) PHI(GRAD)
     TAB 10 2.5,60/11 2.5,45/12 2.5,30
```

Description of the location of the outer edge:

```
LBASE 400 DIS X 5.5(CM) DIS Y 5.5(CM)
          ROT 180(GRAD)
COOR 400 POLAR R(CM) PHI(GRAD)        2
     TAB 30 3,20/31 3,45/32 3,70
```

Description of the location of the remaining nodes:

```
PARC 400 XI 3 ETA 2 BOUND
     RECTA 12,11,10,32,31,30
     TAB 20 1,0.5/21 0.5,0:5/22 0,0.5
```

1- Mesh, 2 - Degrees

Fig. 3.14: Simple examples for the description of a configuration

3.7 Duality of the Degrees of Freedom

3.7.1 Dual Vector Spaces in Physics

The finite element method was first applied in static and dynamic calculations for structures. Here the physical theory of elasticity was made useable for engineering practice through a restricted -- limited to finite elements -- approximation solution of the fundamental theoretical equations. It was however soon recognized that the method did not need to be restricted to elasticity calculations [Argyris 69,72]. It is applicable to all physical areas in which the theoretical foundations are described by two dual physical parameters, $\mathbf{u}$, $\mathbf{v}$. Let U and V be two well-behaved vector spaces with equal finite dimensions and $\mathbf{u} \in U$, $\mathbf{v} \in V$. Then the duality of U and V is described by a scalar product

$$E = \beta (\mathbf{u}, \mathbf{v}) \tag{3.21}$$

which is positive definite, thus:

$$E > 0 \text{ for } \mathbf{u}, \mathbf{v} \neq \mathbf{0} \quad . \tag{3.22}$$

In elasticity theory the physical meaning of energy can be attributed to the scalar product:

$$E = \int_V \varepsilon^T \sigma \, dV = 2 \bar{U} \tag{3.23}$$

with the dual vector quantities

$$\varepsilon = \{\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \varepsilon_{xy}, \varepsilon_{xz}, \varepsilon_{yz}\}: \text{ strain vector}$$

$$\sigma = \{\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{xz}, \sigma_{yz}\}: \text{ stress vector}$$

and U as deformation energy. Besides this the vector spaces U and V

are related to each other such that V is the dual space of U, which includes all linear combinations of the form

$$v = \varphi(u) \tag{3.24}$$

Physically this has the significance of the existence of a constitutive equation in the theoretical area which interrelates the two dual parameters. If one uses this relation to depict the scalar product this yields the quadratic form

/53

$$E = \beta(u, \varphi(u)) > 0 \quad . \tag{3.25}$$

In elasticity theory the constitutive equation is known as Hooke's Law:

$$\sigma = E\varepsilon \tag{3.26}$$

where **E** is the elasticity tensor.

In summary, elasticity possesses the following properties which are important for the application of the finite element method in linear elastostatics:

1. The scalar product of two related dual parameters may be considered energy.

2. The dual parameters are related to each other by a linear constitutive equation.

To cite further examples of linear, static areas of application of the finite element method besides elasticity theory:

- Electrical field with the dual parameters electrical field strength **E** and electrical displacement density **D**. The scalar product is

$$E = \int_V \mathbf{D}^T \mathbf{E} \, dV$$

and the applicable constitutive equation is

$$\mathbf{D} = \varepsilon \mathbf{E}$$

where ε is electrical susceptibility.

- Magnetic field with the dual quantities of magnetic field strength **H** and magnetic field density **B**. The scalar product is /54

$$E = \int_V \mathbf{B}^T \mathbf{H} \, dV$$

and the applicable constitutive equation is

$$\mathbf{B} = \mu \mathbf{H}$$

where μ is magnetic permeability.

It is characteristic of all these areas of application that the components of the model description are always the same. From this results the uniformity of the finite element method, which is most necessary for its wide practical application.

3.7.2 Duality Relationships

The duality of physical quantities manifests itself also in the specification of a measurement procedure; specifically, all measurement variables in a physical application are divided into so-called flow or cross-section variables and potential or reference

point variables. The first name reflects the physical significance, the second, the measuring procedure. Thus, flow properties are measured with the use of a cross-section and determination of the throughput, while potential quantities are measured between two reference points as a potential difference. In elasticity theory, force and stress are flow properties; displacement and strain, potential quantities.

Through such a classification one obtains several pairs of dual variables in a theoretical area. If one compares two such pairs **u**, **v** and **u′**, **v′**, one can depict the relationships between these quantities in a graph (Fig. 3.15). Each node of the graph represents a quantity with which the potential quantities are coordinated on the left side and the flux quantities on the right side (or vice versa). Each edge represents a linear relationship.

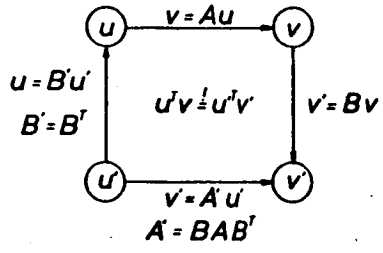


Fig. 3.15: Relationships between pairs of dual quantities.

Between dual quantities exists a constitutive equation and between /55 similar quantities exists a compatibility requirement. According to the classification, one has one compatibility requirement for flow variables and another for potential variables. These are not independent of each other and represent a dual descriptive pair. For this, applies first of all the constancy of the scalar product (i.e. the energy)

$$\mathbf{u}^T\mathbf{v} = \mathbf{u}'^T\mathbf{v}' \qquad (3.27)$$

for discrete quantities and with

$$\begin{aligned} \varphi &: u = Bu' \\ \varphi^* &: v' = Bv \end{aligned} \tag{3.28}$$

results

$$u'^T B^T v = u'^T Bv$$

from which follows

$$B' = B^T \tag{3.29}$$

Therefore φ is the compatibility requirement adjunct to φ^* (compare [Kowalsky 79]). The compatibility requirement always exists and is unambiguous. The constancy of the scalar product used here signifies physically the independence of the energy from the quantities which define it, as long as such quantites are mutually compatible. From this may be derived a relationship between constitutive relations of the two pairs of quantities. Once again

$$u^T v = u'^T v'$$

/56

or

$$u^T Au = u'^T A'u'$$

and with

$$u = B'u'$$

then

$$u'^T B'^T AB'u = u'^T A'u'$$

from which

$$\mathbf{A}' = \mathbf{B}'^{T}\mathbf{A}\mathbf{B}' \tag{3.30}$$

or with Eq. (3.29)

$$\mathbf{A}' = \mathbf{B}\mathbf{A}\mathbf{B}^{T} \tag{3.30a}$$

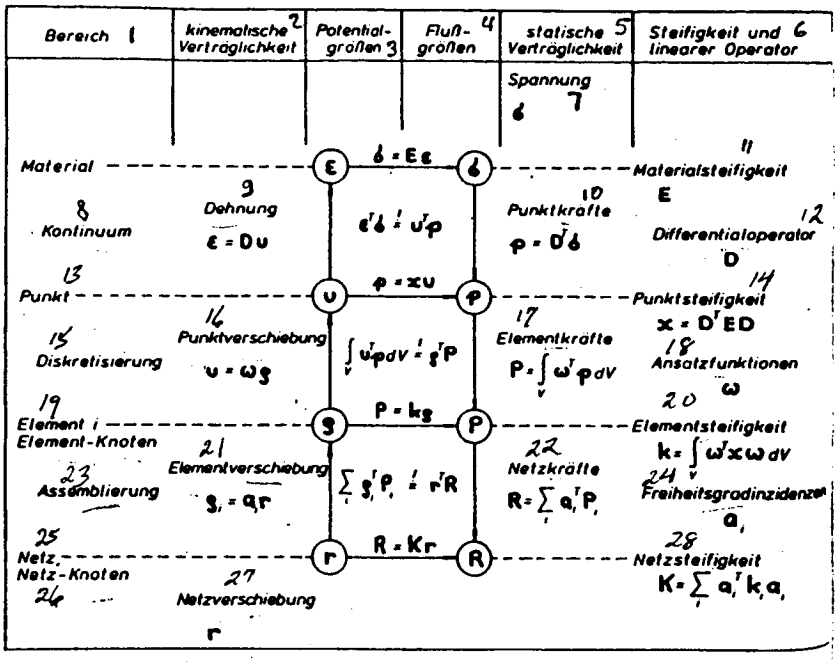
and thus

$$\mathbf{v}' = \mathbf{B}\mathbf{A}\mathbf{B}^{T}\mathbf{u}' \tag{3.31}$$

The relations of Eqs. (3.29) and (3.30) are applicable to all pairs of dual parameters and they make possible the transition from one pair of quantities to another as a consequence of the linear relations.

3.7.3 Displacement Method

The duality relationships can be applied in a multistage manner to a dual parameter pair. In Fig. 3.16 the resulting graph is shown as a duality ladder for the displacement method of finite elements in structural calculations. While the uppermost rung is determined by the physics of the continuum, the second rung depicts the most important step in the derivation of the finite element method. Thus from the continuous displacement field $\mathbf{u}$ the node of discrete elements is transferred to the displacement $\boldsymbol{\rho}$. This step of discretization has contributed to the development of the method, above all by the principle of virtual work [Argyris 54,56,57]. The dual parameter /57 pair $\mathbf{s},\boldsymbol{\rho}$ is designated the element degrees of freedom. These are related to the mesh degrees of freedom, $\mathbf{r}$, $\mathbf{R}$ on the third rung of the ladder. While in the derivation of the displacement method the static compatibility is deduced from the kinematic compatibility, the opposite is true of the dual force method [Argryis 54,56,57]. The force method may be depicted by a dual ladder by which all edges of the graph have the opposite orientation. The existence of two dual methods follows from the principle of dualization [Kowalsky 79].



1 - region, 2 - kinematic compatibility, 3 - potential variables, 4 - flow variables, 5 - static compatibility, 6 - stiffness and linear operator, 7 - stress, 8 - continuum, 9 - strain, 10 - point forces, 11 - material stiffness, 12 - differential operator, 13 - point, 14 - point stiffness, 15 - discretization, 16 - point displacement, 17 - element forces, 18 - statement function19-element nodes, 20 - element stiffness, 21 - element displacement, 22 - mesh forces, 23 - assembly, 24 - incidences of degrees of freedom, 25 - mesh, 26 - mesh node, 27 - mesh displacement, 28 - mesh stiffness

Fig. 3.16: Duality ladder for the displacement method in structural calculations.

/58

3.8 Kinematic Boundary Conditions

3.8.1 Incidences of Degrees of Freedom

In the dual vector spaces of the degrees of freedom the coupling of the meshes is brought about by incidences of degrees of freedom. For incident degrees of freedom the degrees of freedom of the lower mesh are designate external degrees of freedom and the corresponding upper mesh degrees of freedom are designated actual degrees of freedom. The quantity of degrees of freedom A of an upper mesh consists of the union of all external degrees of freedom E_i of each coupled lower mesh i:

$$A = \bigcup_i E_i \quad . \tag{3.32}$$

This condition is expressed in terms of matrices as

$$r_i = a_i r \tag{3.33}$$

by which $r_i \in E_i$. The total degrees of freedom of the upper mesh is designated by r, and a_i denotes the incidence of degrees of freedom of the lower mesh (compare [Schrem 78a]). Depending on the mesh type, each node of a connecting mesh carries the same set of degrees of freedom which can be actual. One calls the quantity of all degrees of freedom of a mesh the quantity of potential degrees of freedom Π, out of which all the actual degrees of freedom must be taken, i.e.

$$A \subseteq \Pi \quad . \tag{3.34}$$

The following apply for the distribution of external degrees of freedom in the mesh tree:

- The main mesh has no external degree of freedom.
- Partial meshes must have external degrees of freedom, otherwise the lower mesh tree contributes nothing to the physical model.
- Elementary meshes normally have only external degrees of freedom, because the choice of a certain element type is based precisely on its degree of freedom.

/59

Degrees of freedom are transferred upwards by means of incidences into each sucessive upper mesh until they no are longer arranged as external degrees of freedom. This release of the degrees of freedom can occur in all meshes, but must occur in the main mesh at the latest. The correlation with the node incidences according to the principle of node-connected degrees of freedom (Chapter 3.2) is based

on the fact that exclusively incident nodes can have degrees of freedom. Thus,if a lower mesh node has a degree of freedom, it must also be incident. Conversely, a lower mesh node must also have a degree of freedom if it is incident; otherwise,the coupling in the visualization space must occur differently from in the dual vector spaces of the degrees of freedom. Incident nodes take up the same point in the visualization space. For incident degrees of freedom apply:

- o The displacement values are equal: kinematic compatibility
- o The sum of the lower mesh node forces is equal to the force on the upper mesh node: static compatibility.

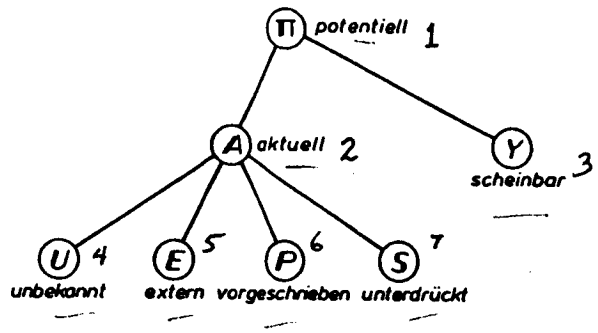
This is the physical significance of the assembly in Fig. 3.14.

3.8.2 Classes of Degrees of Freedom

All degrees of freedom occurring in a connecting mesh are included in the quantity of potential degrees of freedom. Their canonical order is defined by the node order and the order of all degrees of freedom in a node, the latter being determined by the mesh type.

A division of the potential degrees of freedom into classes of degrees of freedom serves to divide the dual vector spaces into distinct lower spaces and hence also serves the modular treatment of the boundary conditions [Schrem 78a]. Here the actual degrees /60 of freedom play a central role (compare Fig. 3.17). They include all those degrees of freedom which can be traced over incidences back to element degrees of freedom. In contrast to these are the apparent degrees of freedom γ' which denote the unused potential degrees of freedom.

$$\gamma' = \pi \setminus A \qquad (3.35)$$



1 - potential, 2 - actual, 3 - apparent, 4 - unknown, 5 - external, 6 - prescribed, 7 - suppressed.

Fig. 3.17: Classification of the degrees of freedom

A node whose potential degrees of freedom are all apparent degrees of freedom is called an apparent node. The quantity of actual degrees of freedom includes the following subquantities (Fig. 3.17):

- o External degrees of freedom $\mathbf{E}$: They serve the coupling into the upper mesh and may not be described for the main mesh.

- o Internal degrees of freedom: the dual character of the degree of freedom requires its division into:

  a) Unknown degrees of freedom $\mathbf{u}$: Their displacement values are a priori unknown. Their quantity determines the dimensions of the calculation tasks in each mesh.

b) Given degrees of freedom: For them a displacement value is known a priori. Through this,results in the model a constraint; thus,such given conditions are called kinematic constraints. The constraint results from the quantity of degrees of freedom being reduced due to projection into a displacement space of smaller dimensions. That also /61 reduces the dimension of the constitutive equation to be solved

$$\mathbf{R} = \mathbf{K}\mathbf{r} \tag{3.36}$$

Here one differentiates:

1. Prescribed degrees of freedom $\mathbf{P}$: For them an arbitrary value is provided for, for which reason these measurement values are counted among the load-dependent data of the model description (compare Chapter 3.10).

2. Suppressed degrees of freedom $\mathbf{S}$: For them the displacement value of zero is assigned. They make no contribution to the displacement field of a model and are not considered in the calculation.

Thus,there result five classes of degrees of freedom and it applies that:

$$\begin{aligned} \mathbf{E} \cup \mathbf{U} \cup \mathbf{P} \cup \mathbf{S} \cup \mathbf{Y} &= \mathbf{M} \\ \mathbf{E} \cap \mathbf{U} \cap \mathbf{P} \cap \mathbf{S} \cap \mathbf{Y} &= \mathbf{0} \end{aligned} \tag{3.37}$$

In the method of writing matrices the selection of displacement values of a class K is achieved with the class selection matrix $\mathbf{b}_K$ (compare [Schrem 78a]):

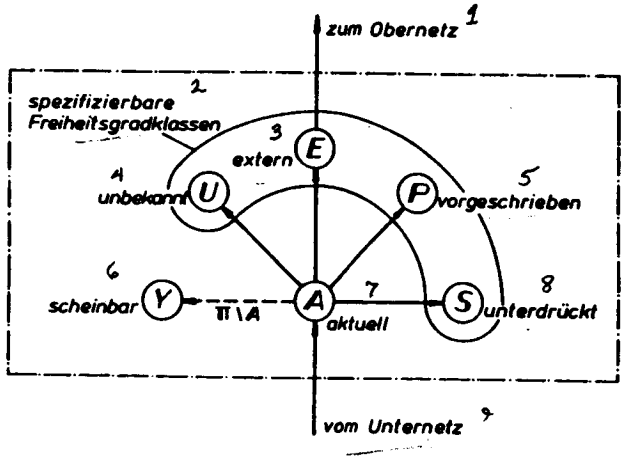
$$r_K = b_K^T \, r \qquad (3.38)$$

where the potential degrees of freedom of a mesh are shown by r (compare Eq. (3.33)). In the model description the class Y is never specified. A degree of freedom is assigned to the class U if it either is designated as such or if it is an actual degree of freedom associated with any class. The description of all the other class affiliations must be done by the user. Here the following rule is to be observed

/62

> The model must be kinematically definite, i.e. rigid body movements of the entire model are to be excluded.

That can happen through a suitable orientation of the structure with the help of suppressed degrees of freedom.



1 - to upper mesh, 2 - specified classes of degrees of freedom, 3 - external, 4 - unknown, 5 - prescribed, 6 - apparent, 7 - actual, 8 - suppressed. 9- to lower mesh

Fig. 3.18: Plan for classification of degrees of freedom in a partial mesh.

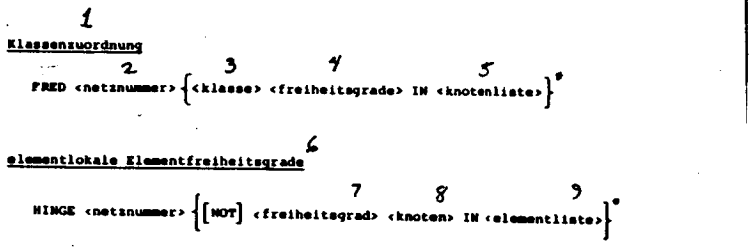
3.8.3 Element Degrees of Freedom

In elementary meshes the canonical order of the degrees of freedom results from the order of the elements, the node order in the element mesh and the element type-dependent quantity of degrees of freedom on each node. Commonly the following simple relationships apply for the quantities of degrees of freedom in elementary meshes:

$$\Pi = A = E \quad . \tag{3.39}$$

/63

A change in these relationships results from the introduction of element-localized element degrees of freedom. Through this the number of degrees of freedom is reduced. This serves e.g. the simulation of joints between elements. Fig. 3.19 shows how commands for description of joints and degrees of freedom appear. The two small examples in Fig. 3.20 illustrate the use of the commands.

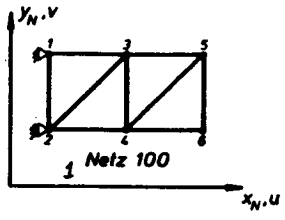


1 - class coordination, 2 - mesh number, 3 - class, 4 - degrees of freedom, 5 - node list, 6 - element localized degrees of freedom, 7 - degree of freedom, 8 - node, 9 - element list

Fig. 3.19: Commands for the description of kinematic boundary conditions.

Coordination of classes of degrees of freedom:

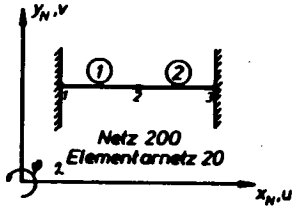
```
FRED 100 SUPP U,V IN 1
         SUPP U   IN 2
```



The remaining degrees of freedom are automatically associated with the "unknown" class.

Jointed connection of two beam elements:

```
FRED 200 SUPP U,V,PHI IN 1,3
HINGE 20 PHI 2 IN 1,2
```



1 - mesh, 2 - elementary mesh

Fig. 3.20: Examples of the description of kinematic boundary conditions.

3.9 Element Data

Characteristic of the finite element method is the uniformity which allows

- o mesh tree
- o topology
- o configuration and
- o boundary conditions

for all element types to be treated in the same way in the model description. A series of properties of the elements cannot be so taken in, and these are all those which outside the configuration are necessary for the calculation of an element (e.g. for the stiffness). These elements concern:

- o geometry and
- o material of the model

/65

and depend on the element type. By geometry is understood

- o Cross-section data on rod and beam elements such as area, moment of inertia, torsional moment of inertia, shear areas, etc.
- o Cross-sectional data on membrane,shell, and plate elements as thickness, distribution of layers, etc.

The key word material unites all parameters which describe the material behavior, e.g.

- o material stiffness, including a possibly necessary reference direction system
- o density
- o coefficient of thermal expansion, etc.

An element type can have several sets of such element data, which can be determined depending on the resources available for the element calculation, e.g.

- o isotropic or anisotropic material stiffness,
- o solid beam or beam with open,thin-walled construction, etc.

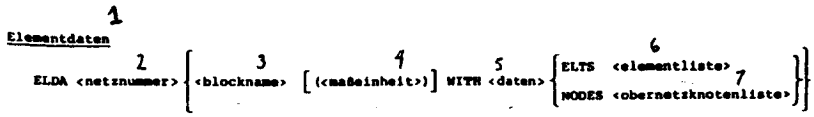
Which set is under considerationis determined by the so-called model type. This encompasses the variety of elements of a type. The selection of the model type prevails over the model description. Here is to be heeded:

- > For all elements of an elementary mesh applies the same model type.

The model type serves for one thing the guidance of the input data with respect to storage and checking the completeness of the model description. For another thing,it directs the element calculation in regard to its scope and thus its efficiency. The principle of unity of model and mesh tree demands then that to every given element type the corresponding model type is also determined. Thus,the user determines exactly which physical phenomena he plans to investigate.

For the description of element data, two fundamentally /66
different possibilities are given. One possibility occurs on elementary mesh planes by statement of the element number. This is the usual way. In the other way the description takes place on connecting mesh planes. Here,the data are allotted to certain nodes and coordinated with the elements concerned by means of node incidences. This possibility is excluded if the element datum in question is not equal in all elements concerned, or if elements of different types are coupled in this connecting mesh node, but the datum is not contained in all model types concerned. Fig. 3.21 shows the command for description of the element data. An example of its application is found in Appendix A.

In the description of element data, dimension-dependent variables appear, as already seen to some degree in the configuration and the combined constraints. All their units must be compatible, in order to be able to interpret correctly the results of the calculation. This compatibility can be left the responsibility of the user, but for the description it is significantly more convenient to ensure automatically the compatibility of input measurement numbersby specifying measurements. This occurs by the conversion of all quantities to the MKSA system of units [DIN 1301]. Thus,the user has a variety of possibilities for input of data.



1 - element data, 2 - mesh number, 3 - block name, 4 - mass unit, 5 - data, 6 - element list, 7 - upper mesh node list

Fig. 3.21: Command for description of element data.

/67

3.10 Loading

In the description of loading,all given parameters of the degrees of freedom (displacement and forces) are specified. Thus dual parameters in a node cannot both be specified at the same time. In loading, one differentiates between

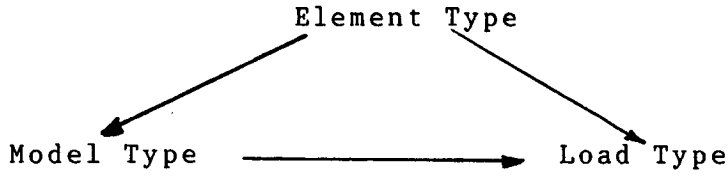
- Node loads on planes of the connecting mesh. In this category fall node forces and node displacements for the prescribed degree of freedom and for the prescribed portion of the dependent degree of freedom.

- Element loads on elementary mesh planes. These include:
  1. Distributed loads:
     a) Volume loads from accelaration such as weight and centrifugal force
     b) Surface loads from pressure
     c) Line loads
  2. Initial loads
     a) Initial strains (e.g. from temperature)

As the model type determines a set of element data, so is a load type provided for, which chooses those loads for a certain element type which are needed for a model description. Which load types are possible is determined by the element type.

> For all elements of an elementary mesh applies the same load type.

Out of the divided element loads, kinematic equivalent node forces are determined [Argyris 65]. These are transmitted to the upper mesh via incidences and there they are added into the node loads. Since certain element data are important for this calculation, the choice of load type also depends on the model type. One must also pay attention to the following schema of dependencies:



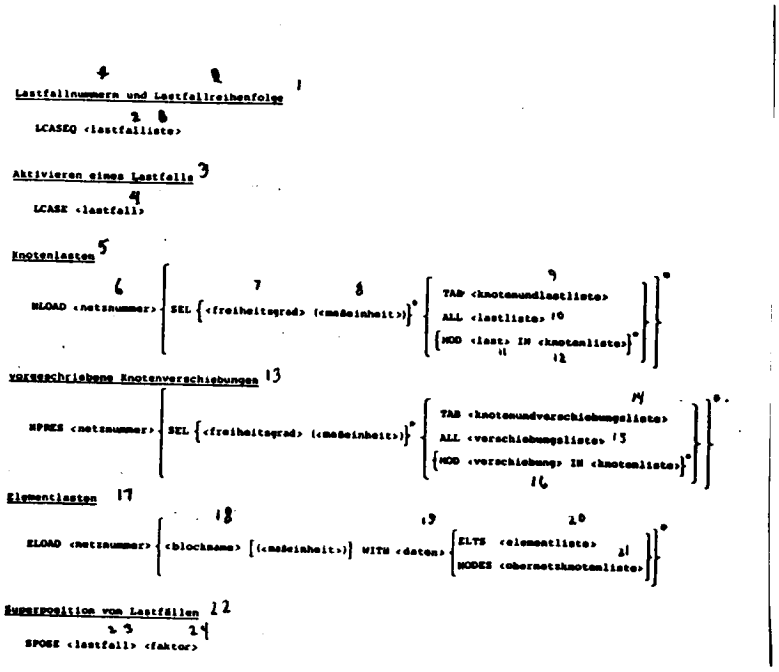
The load type serves firstly to guide the element load input and /68 secondly for the guidance of the calculation of the initial loads and the kinematically equivalent node loads. According to the procedure for input of element data, the possibility of description on planes of elementary meshes or connecting meshes also exists with element loads.

A definite combination of node and element loads for a model is called a loading case. Each model allows a fundamentally free selection of loading cases. Each loading case is identified by means of a loading case number. As with nodes and elements, the user determines the order of his external loading case numbers, which correspond to ordinal internal loading case numbers. In this way the user is free in his choices of identifiers for loading cases. Single loading cases can represent combinations of other loading cases. The quantity of loading cases is a global property of the model. All meshes possess the same number of loading cases. According to the principle of the unity of the model and mesh tree, this number must be determined during the mesh tree description. The same is true for the load type of an elementary mesh. According to the dependency schema

shown above, and in order to fix all global model properties from the beginning it will be agreed:

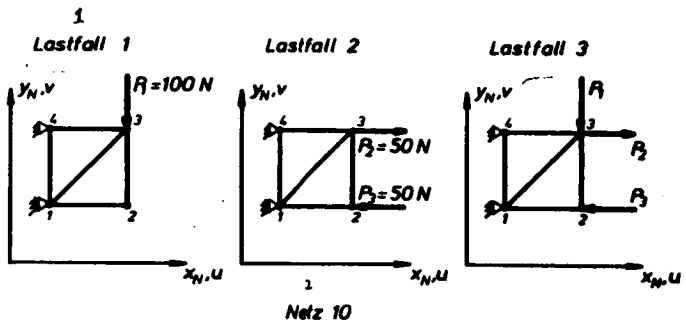
> One load type applies for all loading cases.

As with the model type, the user determines the physical phenomena to be studied through the choice of the loading type. This serves a precise idealization which always must precede a model description. This means that the user must consider precisely in what condition his model is. These considerations affect all the properties which are to be fixed in the mesh tree description. In reference to the physics of the calculation assignments, the types of mesh elements, element models, and element loads are decisive. The well-considered choice of these types makes their alteration unnecessary in the further course of the model description. The principle of unity of model and mesh tree is thus tightly connected with a properly sequenced way of working on the part of the user: first, the complete idealization of the model; then its description. The commands for description of /69
loading are summarized in Fig. 3.22. An explanatory example with three loading cases is shown in Fig. 3.23.



1 - loading case numbers and loading case order, 2 - loading case list, 3 - activation of a loading case, 4 - loading case, 5 - node loads, 6 - mesh number, 7 - degree of freedom, 8 - unit of measurement, 9 - node and load list, 10 - load list, 11 - load, 12 - node list, 13 - prescribed node displacements, 14 - node and displacement list, 15 - displacement list, 16 - displacement, 17 - element load, 18 - block name, 19 - data, 20 - element list, 21 - upper mesh node list, 22 - superposition of loading cases, 23 - loading case, 24 - factor

Fig. 3.22: Commands for the description of the loading.



3

```
Beschreibung:  LCASEQ 1,2,3

               LCASE 1
               NLOAD 10 SEL V(N) TAB 3 -100

               LCASE 2
               NLOAD 10 SEL U(N) TAB 2 -50 / 3 50

               Superposition der Lastfälle 1 und 2   4
               LCASE 3
               SPOSE 1 1.0
               SPOSE 2 1.0
```

1 - loading case, 2 - mesh, 3 - description, 4 - superposition of loading cases 1 and 2.

Fig. 3.23: Simple example for description of node loads.

3.11 Calculation

Part of the total purpose of a system for modelling is to place in readiness all matrices with which the core system can perform the FE calculation. If one takes as a basis the solution algorithm as it is implemented in the program system ASKA for a linear static analysis [ASKA 71], then the following matrices are needed:

/71

- for all lower meshes j of a connecting mesh: the matrices of incidences of degrees of freedom a_j (compare Eq. 3.33).
- for all upper meshes:
  a) the class selection matrices b_U, b_E, b_P, b_S (compare Eq. 3.38). Thus, result the class-specific matrices of incidences of degrees of freedom in the corresponding lower mesh

$$\begin{aligned} a_{Uj} &= a_j b_U \\ a_{Ej} &= a_j B_E \\ &\text{etc.,} \end{aligned} \qquad (3.40)$$

  b) the node loads R′ for all loading cases,
  c) the prescribed displacements r'_p for all loading cases.

Through the concept of the satellite system (compare Chapter 2.1), in particular the entire element calculation is removed from the core system. To that part of the model description belong:

1. the checking of all supplied data of each single element,
2. the construction of element stiffness matrices k_{ei} for all elements i, so that for the stiffness matrix of an elementary mesh results

$$k_j = \left[k_{ei}\right] \quad . \qquad (3.41)$$

3. the assembly of the element forces Q_{ei} based on initial loads j_{ei} and distributed element loads q_{ei} for all elements i and all loading cases so that it results in

$$Q_j = \left[q_{ei} - j_{ei}\right] \quad . \qquad (3.42)$$

With these preconditions, all necessary steps for solution of the calculation task can be performed in the core system. These are performed in Appendix B.

4. Modularization of the Program and the Data /72

4.1 Segments

4.1.1 Concepts

A definite application of the finite element method -- called a project -- always begins with that part of the satellite system concerned with the model description. Thus, in the model description the user first comes into contact with the VFE-machine. All available model data are then stored in a single special project data base and processed (see Chapter 5) by the data processing system (DVS). For the following illustration of the modularity of the program and the data, it suffices here first to assume that so-called data modules serve as storage units in the project data base. In each of these modules is stored all the data of a model object which describe a particular property, e.g. all node incidences of a lower mesh with its upper mesh, or all node coordinates of a mesh.

The components of the model description could be depicted in easy-to-follow form by the division into several steps, one following the other. Following the principle of modularization, this division should also be carried out for the programming system, the data module, and the user input. The pertinent concept is called a segment. Each segment represents the sum of the following decisions:

- The user should have in mind an easily-overviewed and homogeneous part of the model description and thus be able to abstract from the other parts as extensively as possible.

- A segment is a collection of tools for the convenient description of model data. In using these tools, the user should need no knowledge about the data modules used internally in the program.

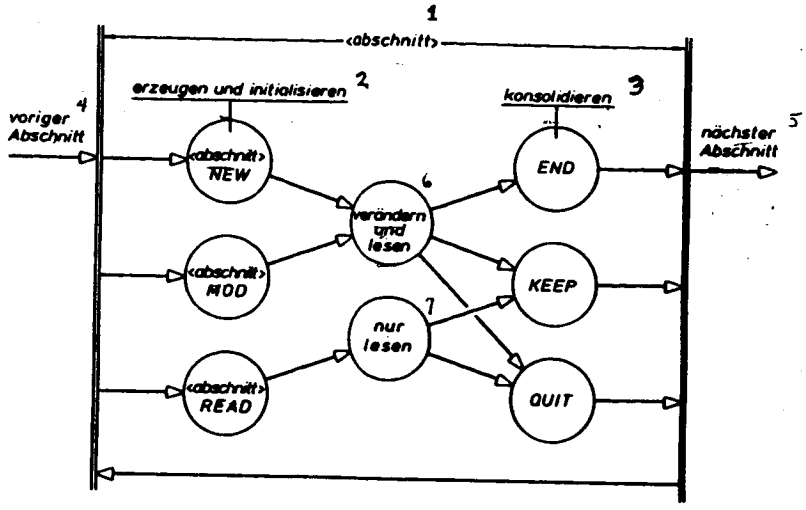
- The fundamental operations of creating and initializing (or destroying) as well as changing data are allocated exactly one segment. The coupling of allocated segments is realized solely by read access to data already described. /73

Thus each segment depicts one functional unit in the view of the user. For the elastostatic model description result the following segments (with their respective names):

A1	TREE	mesh tree
A2	TOPO	topology
A3	CONFIG	configuration
A4	BOUND	kinematic boundary conditions
A5	ELDA	element data
A6	LOAD	loading case
A7	ELCA	element calculation

Each segment is shown to the user in the following phases (compare Fig. 4.1):

1. Activation of the segment: This is called up by the user either as a first call (NEW), continuation call (MOD) or reading call (READ). The differentiation of first and continuation calls serves the strict separation of creation and alteration of data modules (compare to the separation principle of [Schrem 78a]). This separation is closely correlated with a static structure of the project data base, the details of which still need to be carefully considered. In the first call all segment-specific data modules are assigned the state "existing and empty". The first call and all continuation calls can change the content of the data module. The types of calls serve to define the access rights:



1 - segment, 2 - creation and initialization, 3 - consolidation, 4 - previous segment, 5 - next segment, 6 - change and read, 7 - read only

Fig. 4.1: User calling diagram of a segment.

- o NEW : create, initialize (or destroy), change, read
- o MOD : change, read
- o READ : read

2. Work with the segment: The user effects the changing or reading of the segment-specific data module. The scope of the possible operations is guided by the access rights set forth in the first phase of this segment.

3. Deactivation of a segment: This is undertaken by the user either to interrupt the work (KEEP, QUIT) or at the close of the work (END). An interruption can be desired to activate another segment (KEEP) or to end the work with the system (QUIT), which leads to a return to the operating system level. In both cases, it is ensured that the current data state is not disturbed and, after a new activation (by MOD or READ) the previous conditions are found again exactly. If the user believes to have described all data for a segment, he will bring about a so-called consolidation (END). This central concept serves the verification of the completeness and compatibility of the inputted model data. Thus, the user is reassured from

segment to segment of the correctness of the model description. The consolidation is called successful if no completeness or incompatibilities are found; if this is not the case, it is called unsuccessful. Through multiple /75 activation of a segment and subsequent correction of the data content, the user can always ensure a successful consolidation. These circumstances are summarized in the principle of consolidation:

> Each model description must be successfully consolidated in all segments. Only then may it be called successful.

If a segment is successfully consolidated, a new activation with a first call is excluded, because this would cause all pertinent data modules to be newly created, which would lead to destruction of the consolidated state of the data base. Besides the desired verification, a broadening of the data base can be done in the consolidation (compare the consolidation of the node coordinates, Chapter 3.6.2). Thus, a consolidation can not be performed for a read-only segment.

Some further consequences of the consolidation principle are indicated: For one thing, all consolidations must be very carefully planned in order to expose all sources of error (compare Chapter 3). For another thing, a considerable effort is connected with all consolidations, both in view of the programming involved as well as the computation time. This is balanced by the certainty of the user of having given a correct model description in the framework of the verifiable conditions. This is of great significance for the subsequent calculation of the model and evaluation of the results.

4.1.2 Control

All segments of the model description may be judged with respect to state and handling of their model data according to the following viewpoints:

1. Which segments must be processed and consolidated before the segment in question, since the access to already existing data is necessary for the creation of new data?

2. Which access rights to the project data base exist for the statement in question?

3. Are the data of the segment in question fully consolidated?

Since the correct consideration of these viewpoints is decisive for successful work with the system, it suggests itself to give the user a helping hand to save him the necessity of bookkeeping the state of his model description.

The first step in that direction consists in the introduction of a higher-ranked control segment which is always active at the beginning of system use and which knows and can call every segment demanded by the user. Each segment is thus informed of the access rights to the data modules of the project data base (whether NEW, or MOD, or READ). Afterwards, the segment takes the sole control of the model description until the user interrupts or ends with a consolidation of further input to this segment. Following this, the control segment is active again, and is informed of the conclusion of the segment (whether QUIT, KEEP, or END; whether successfully consolidated or not).

The second step consists in the creation of a segment control list which represents the static call diagram of all segments. This is a component of the project data base and in it takes place the

bookkeeping for the consolidation of all segments. Thus,it is determined in which order the segments should be called if the consolidation occurs successfully. The actual dynamic calling /77
order can deviate considerably from this and include many iterations. In the elastostatic model description,the static calling order agrees with the sequence -- A1 through A7 -- given above (Chapter 4.1.1).

The third and final step to segment control lies in the development of a functional unit, whose inner state is represented by the segment control list, and which decides according to this list whether a segment can be called or not. Thus,the control segment is placed in the position to react to the demands on a segment from the user according to the continuation of the model description. The entrances to this functional unit are consistent with the control possibilities of the user (compare Fig. 4.2):

1. First calling of a segment

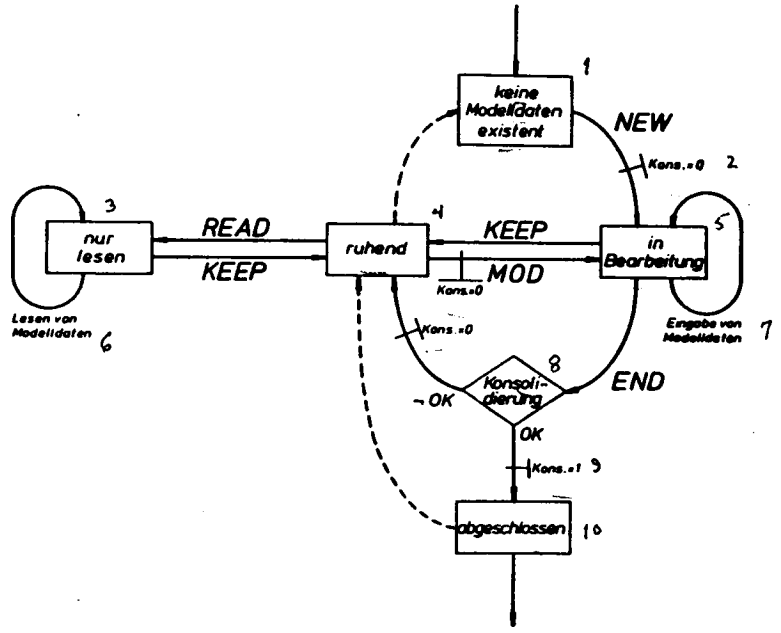
   SECNEW (name[3], permissibility)

   'name[3]' : segment name, up to 12 symbols long
   (FORTRAN format 3A)

   'permissibility' = 0 : no
   > 0 : yes

   This function determines the rank of the segment named. It examines whether all lower-ranked functions have been successfully consolidated. If that is the case, the user receives a message. Then the function deletes all entries for the requested function and all higher ranked segments, if such entries are available. Thus,it is hindered that the determined compatibility of the model data of a segment be destroyed by subsequent changes in the data. This has the consequence that this higher ranked segment must be

once again consolidated in every case. If the demanded segment was already successfully consolidated, then the calling of the function is not permissable, and the user receives a message. Thus,it is hindered that existing data can be destroyed by mistake. If the calling of the segment is permitted, then, after the call, the corresponding actual parameter receives the segment number which in the control segment can be used as a jump variable for a multiple branching. /78



1 - no model data extant, 2 - consolidation = 0, 3 - read only, 4 - dormant, 5 - in process, 6 - reading of model data, 7 - input of model data, 8 - consolidation, 9 - consolidation = 1, 10 - closed.

Fig. 4.2: Flow chart of a segment

2. Continuation calls of a segment

SECMOD (name[3], permissibility)

This function differs from the previous one only in that the change of an already unsuccessfully consolidated segment is permissible, if it is also confirmed with a warning notice to the user. Thus,the difference is preserved between correct model description and correct model calculation.

/79

3. Read calls to a segment

   SECRED (name[3], permissibility)

   Such a call is always permissible if the segment is carried in the control list. The function does not change the control list.

4. Interruption of the segment

   SECKEP (name[3])

   The segment must be known but the function does not change the state of the control list. This function is also called from the control segment if the consolidation was not successful.

5. Ending of a segment

   SECEND (name[3])

   After the release of the segment,the success of the consolidation is retained in the segment control list. This occurs, however, only on the condition that all lower ranked segments were successfully consolidated, because otherwise the results will not necessarily be based on irrefutable model data.

Changes of the calling order affect exclusively the segment control list. The state diagram of the functional unit corresponds to Fig. 4.2. For the user, an additional function is provided for:

6. Listing of the state of the model description

   SECLIS

   With this, the user receives the momentary state of the segment control list in easy-to-see form on his output device.

/80

4.2 Contexts

4.2.1 Concepts

In the broadening of a concept already introduced (Chapter 3.3) the principle of unity of model, mesh tree, and project data base is established for the storage of model data:

> All data for a model are stored in one data base, and in one data base are stored the data for at most one model.

This principle forms the framework for the application of the following concepts.

All data modules in the project data base are identified by name. A table of symbols (compare [Knuth 68]) makes possible the correct depiction of names in addresses within the data base. From the principle named results now the necessity to have data modules of the same kind for all meshes in a mesh tree. The processing of the same kind of data modules can then occur truly simply and uniformly, if they are accessible under the same name. In order to ensure the unambiguity of the coordination of names and data modules, all data modules of a mesh are combined under a context (compare [Schrem 78a]),

the so-called mesh context. This is identified with one name as well. This context name corresponds to the respective internal mesh number. The contexts are further implemented as symbol tables which assign the context names to addresses which refer to the symbol table for the data module of every individual context. Therefore the unambiguity of the name must be demanded on one hand for the meshes and on the other hand for the data modules within a context.

Which data module is called by a certain name depends on the actual context, i.e. each context requires activation before the data modules within it can be processed. The data processing system (DVS) places two pairs of functions at one's disposal, in order to create and to activate contexts out of the project data base (see also the flowchart in Fig. 4.3: /81

1. Creation or destruction of an empty context

   NEWCTX (namctx)
   DELCTX (namctx)

   According to the description of the mesh tree, a context is created for each mesh. Through this, a context structure is imprinted which remains unchanged for the entire model description.

2. Activation or deactivation of a context

   USECTX (namctx)
   RELCTX (namctx)

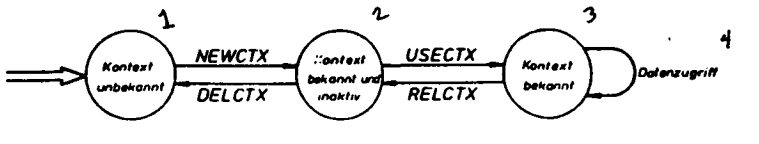
   Since for any given point in time only one context can be active, the deactivation of the first context must precede the activation of the second.

In addition, an additional interrogatory function is available for the name of the current mesh:

ASKCTX (namctx, iex)

The name of the active context is thereby given (iex = 1) or no context is active (iex = 0).

For the input of mesh-independent model data such as the segment control list, beyond the mesh contexts one needs also a so-called basic context. This is always active and allows at any time the access to the data stored in it. In order to ensure the unambiguity of the name in the basic context and each desired active context, it is agreed upon that the names of the data modules in the basic context must always begin with a special symbol (the $ sign). Thus it is already determined by the name whether a particular data module is in the basic context or in the active context. /82



1 - context unknown, 2 - context known and inactive, 3 - context known, 4 - data access

Fig. 4.3: Flow chart for data access by means of contexts.

Since always only one mesh context can be active, there is at first no possibility to simultaneously process two data modules of different meshes. But exactly this is necessary by all assembly precursors by which data from lower meshes are placed into a common upper mesh (compare the consolidation of the node coordinates, Chapter 3.6.2). The existence of incidence information requires the common access to upper and lower mesh data. This purpose is served by the introduction of the so-called Olympus in the context processing [Schrem 78a]. With

this,it is allowed for exactly one data module out of a mesh context to be accessible from all other mesh contexts. In order to avoid confusion of names, the data modules must also have an Olympian name which exists in no context. This context is implemented in the data processing system by a pair of functions (see also the flow chart in Fig. 4.4):

1. Raising a data module into Olympus

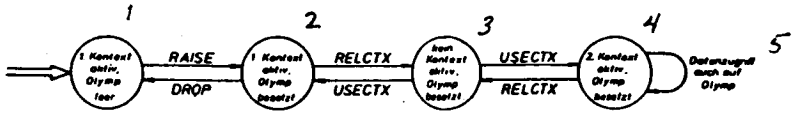
   RAISE (namdat, namolm)

   It must be an active mesh context, out of which the given module is raised into Olympus. In Olympus,the data module carries another name and it is not called its original name as long as it occupies Olympus.

2. Dropping a module out of Olympus

   DROP

   The same context, out of which the module was raised to Olympus must again be active. After calling this function, Olympus is empty. The data module raised is afterwards again accessible under its original name.



1 - first context active, Olympus empty, 2 - first context active, Olympus occupied, 3 - no context active, Olympus occupied, 4 - second context active, Olympus occupied, 5 - data access also on Olympus

Fig. 4.4: Flow chart for data access for contexts and Olympus

4.2.2 Mesh Lists /83

The central significance of the mesh tree for the model description is underscored by the fact that its description serves the first segment. The data structure of the mesh lists and the user input agrees with this concept, as Fig. 4.5 shows in an example. The meshes of the mesh trees are arranged in the lists according to the following rule (compare Chapter 3.3):

- Beginning with the step zero all meshes of a step follow one another and that is in the order from right to left as seen when the tree is viewed from the main mesh outward.

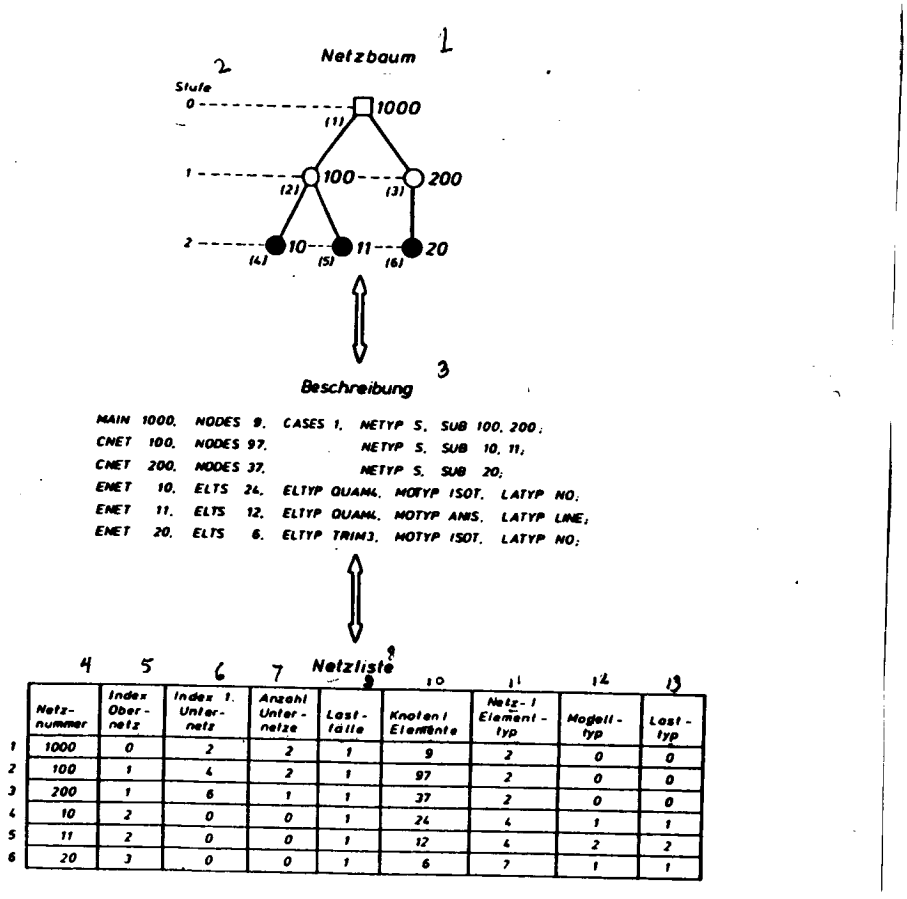
In this way the mesh tree is ordered. The order chosen considers the mesh tree phenomenologically as a family tree, since all sons of a father follow one another in the list. A second, and equivalent order would consist of placing all meshes of a lower tree behind each other in order to depict the concept of a mesh tree as a hierarchy of lower trees.

As super-ordered data structure of the model description the mesh tree is stored in a doubly-interlinked list in the basic context of the project data base [Knuth 68]. It receives, in addition to the linkage information, the following mesh properties (compare Fig. 4.5):

- The context structure of the project data base is determined by the quantity and the numbers of the meshes.

- The structure of the mesh data modules is determined by
  (a) the quantity of nodes and the mesh type of each connecting mesh.
  (b) the quantity of elements and the element type, model type, and load type of each elementary mesh.
  (c) the quantity of loading cases for all meshes.

A change in the mesh properties describes another model. According to the principle of unity of model, mesh tree, and project data base, a new project data base is thus connected with another mesh list. Out of this, results a fully static structure of the project data base. /84 The advantages versus a dynamic concept, by which e.g. the quantity of nodes is not determined at the beginning, lie in a simpler and more efficient data processing and implementation. Last but not least, this design decision has as a consequence fewer and simpler commands for the user.



	Netz-nummer	Index Ober-netz	Index 1. Unter-netz	Anzahl Unter-netze	Last-fälle	Knoten / Elemente	Netz-/ Element-typ	Modell-typ	Last-typ
1	1000	0	2	2	1	9	2	0	0
2	100	1	4	2	1	97	2	0	0
3	200	1	6	1	1	37	2	0	0
4	10	2	0	0	1	24	4	1	1
5	11	2	0	0	1	12	4	2	2
6	20	3	0	0	1	6	7	1	1

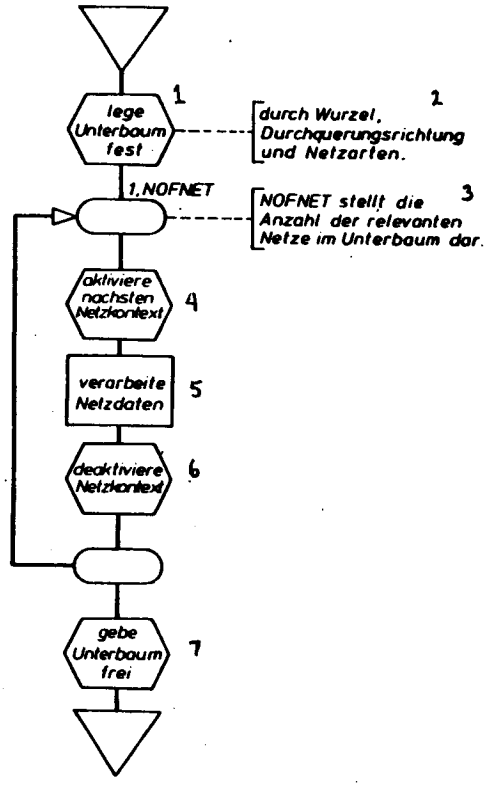
1 - mesh tree, 2 - rank, 3 - description, 4 - mesh number, 5 - index, upper mesh, 6 - index, first lower mesh, 7 - number of lower meshes, 8 - mesh list, 9 - loading cases, 10 - nodes/elements, 11 - mesh/element type, 12 - model type, 13 - load type

Fig. 4.5: Example of the depiction of a mesh tree as a mesh list and its description

C-2

4.2.3 Mesh Tree Traversing

The invention of the mesh tree concept may be traced back to the application of the partial structure technique in model calculations [Argyris 70]. This technique is based on a modularization of the model which was shown to be advantageous in the application of the finite element method to complex models. The general application of the partial structure technique makes use then in the mesh list of suitable means to carry out an automated calculation in all meshes (compare Chapter 3.11).



1 - fix lower tree, 2 - by means of root, traverse direction, and mesh types, 3 - NOFNET depicts the quantity of the relevant meshes in the lower tree, 4 - activate the next mesh context, 5 - process mesh data, 6 - deactivate mesh context, 7 - release lower tree

Fig. 4.6 - General flow chart for mesh tree traverses.

So one speaks of a mesh tree if definite operations for all meshes /86
of a mesh tree are to be carried through in a definite order. The mesh tree may be generalized to all possible lower trees and is completely described by the following properties:

- The root of the lower tree is determined by a mesh number.
- The traverse direction follows the order of the meshes in the mesh list or the reverse order. Graphically, this corresponds to a traverse from top to bottom and left to right or a traverse from bottom to top and right to left (compare Fig. 4.5).
- The input of the mesh types to be considered such as main, partial, or elementary meshes makes possible a selective mesh tree traverse.

If one connects the concepts of mesh list and mesh tree traverse one obtains a general method for the processing of the same data in different mesh contexts. Besides the partial structure technique, the method also finds application for the consolidation phases of the segments of the model description. Fig. 4.6 shows a general program flow chart for the application of mesh tree traversing.

4.2.4 Control of Mesh Contexts

The activation of mesh contexts is a very common task within the software system for the model description. For this reason a functional unit was created which performs the following:

- Abstraction from the mesh list: the mesh tree traverse as well as the activation of definite mesh families is made possible.

- Placing in readiness of functions for activation (or deactivation) of the mesh contexts and for inquiring as to the properties of the active mesh. /87

All functions of the functional unit which change its inner state obey the principle of symmetry, which means that for each of these functions there exists a corresponding inverse function [Schrem 78b]. Such functions act like brackets between which a certain state remains in existence. These circumstances are illustrated in the flow chart in Fig. 4.7. As required by the symmetry of the functions, simple, unified and comprehensible programs arise for processing of the mesh context contents. A consolidated mesh list is the prerequisite for an orderly working of the functional unit. The mesh list will therefore be employed following the mesh tree description in all the segments by which the mesh lists are exclusively read.

The functional unit includes the following pairs of functions:

1. Opening-closing of the functional unit

   NVINIT
   NVTERM

   In between these brackets the project data base must be accessible in order to be able to process. The corresponding state of the functional unit is active-inactive.

2. Mesh tree traverse on-off

   NVMAIN (direction, mesh types, number of meshes)
   NVXMAN

Within these brackets prevails the state:

(a) context name of the main mesh

(b) Traverse direction of the mesh tree:

+1 : in the canonical order of the meshes in the mesh list

-1 : in the reverse order

/88

(c) the mesh types to be considered

0 : all meshes

1 : all connecting meshes

2 : main mesh and elementary meshes

3 : main mesh

4 : partial meshes and elementary meshes

5 : partial meshes

6 : elementary meshes

7 : no mesh

The calling program is given the exact number of the meshes to be considered, in order to be able to control a simple counting loop over these meshes (compare Fig. 4.6). In this loop the following brackets are then opened and closed:

3. Next mesh in the tree active-inactive

NVNODE (number of lower meshes)
NVXNOD

Through this, an internal counting variable is incremented by one and the corresponding mesh is activated in the canonical order. With connecting meshes a loop around all lower meshes can lead to closing of an additional bracket:

4. Next lower mesh active-inactive

   NVSUB (number of lower meshes)
   NVXSUB

   This set of brackets leads to incrementation of a further conunting variable and to activation of the context concerned.

5. Upper mesh active-inactive

   NVSUP (number of upper meshes)
   NVXSUP

   With this the upper mesh can be made accessible to each mesh activated with NVXNODE.

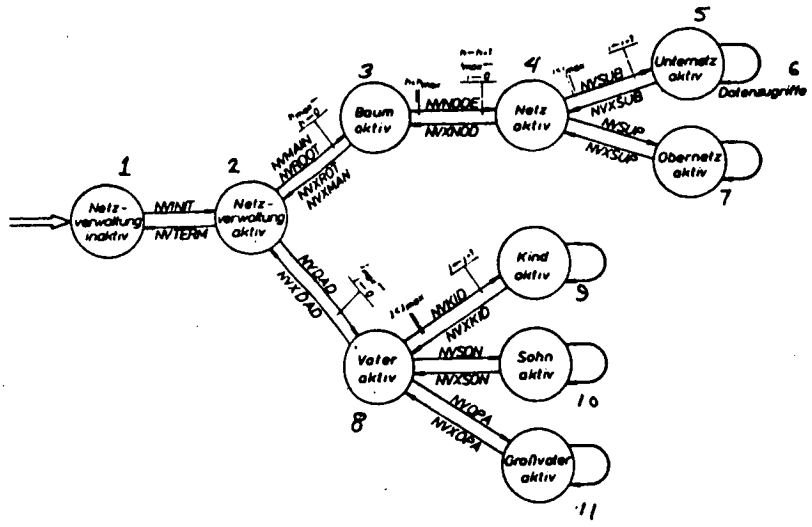
For the traversing of a lower tree, its root must be declared. For this, one must replace the brackets NVMAIN - NVXMAN by means of:

6. Lower tree traverse on-off

   NVROOT (mesh number, direction, mesh type, number of meshes)
   NVXROT

   The name of the lower tree root in internal state is stored as context name. The activation of the meshes and their lower meshes or upper meshes, respectively, occurs analogously to mesh tree traversing.

With this are named all pairs of functions which allow a mesh tree or lower tree to be crossed in a simple manner. This corresponds to a processing of the meshes with respect to the hierarchy of lower trees. By contrast, a second set of functions considers the family, i.e. /90



1 - mesh control inactive, 2 - mesh control active, 3 - tree active, 4 - mesh active, 5 - lower mesh active, 6 - data access, 7 - upper mesh active, 8 - father active, 9 - child active, 10 - son active, 11 - grandfather active.

Fig. 4.7: Flow chart of mesh context control

a mesh with its lower meshes and its upper mesh (or a father with his sons and their grandfather):

7. A certain mesh active-inactive

   NVDAD (mesh number, number of lower meshes)
   NVXDAD

For a further consideration of the family, there are two possibilities: either a certain son is picked out or the children are called in order:

8. A certain lower mesh active-inactive

   NVSON (mesh number, number of lower meshes)
   NVXSON

The state is designated by the context names of father and son.

9. Next lower mesh active-inactive

   NVKID (number of lower meshes)
   NVXKID

   These brackets are called in a loop, which is connected with the incrementing of a counting variable. Here the father must be a connecting mesh in every case.

The consideration of the family is rounded off by a call for the grandfather:

10. Upper mesh active-inactive

    NVOPA (number of upper meshes)
    NVXOPA

In addition to these functions changing the state of the functional unit, the parameters of the momentarily active mesh can be called by the following function:

NVASK (mesh number, upper mesh number, number of lower meshes, number of loading cases, number of nodes/elements, mesh/element type, model type, load type)

/91

Here all current parameters are zero if no mesh context is active.

The significance of the context control for a simple and comprehensible program shall be shown in an example. The assignment

shall be to perform the assembly of stiffness matrices for an entire model and for a desired pair of classes of degrees of freedom (compare Chapter 3.11). The accompanying FORTRAN subprogram has the following form:

```
      SUBROUTINE BKTREE(NAMSA1,NAMSK,NAMSA2,NAMBK)
      DATA NAMOLM /4HOLM /
C
C...  ALLE VERBUNDNETZE IM NETZBAUM VON UNTEN NACH OBEN   (1)
      CALL NVMAIN(-1,1,NOFNET)
C
      DO 100 I=1,NOFNET
         CALL NVNODE(NOFSUB)
C
C...     BRINGE DIE STEIFIGKEITSMATRIX IN DEN OLYMP    (2)
         CALL RAISE(NAMBK,NAMOLM)
C
         DO 50 J=1,NOFSUB
            CALL NVSUB(NSUB)
C
C...        ASSEMBLIERUNG                    (3)
            CALL ATPA(NAMSA1,NAMSK,NAMSA2,NAMOLM)
C
            CALL NVXSUB
   50    CONTINUE
C
C...     LEERE DEN OLYMP   (4)
         CALL DROP
C
         CALL NVXNOD
  100 CONTINUE
C
      CALL NVXMAN
C
      RETURN
      END
```

1 - ALL CONNECTING MESHES IN THE MESH FROM BELOW TO ABOVE,
2 - BRING THE STIFFNESS MATRIX TO OLYMPUS,
3 - ASSEMBLY,
4 - EMPTY OLYMPUS

The processor ATPA performs the assembly of the stiffness of a single lower mesh into its upper mesh. The term "processor" is used here according to the definition used in [Schrem 78a]. In the way shown, all calculated steps of a solution algorithm can be controlled by the partial structure technique, independent of the number of meshes in the model.

5. Management of the Model Data

5.1 General Aspects of Storage Management

All assignments of storage management are performed by the data processing system (DVS), which represents a subsystem of the VFE-machine. For an ordered management it provides for a working memory storage space as well as background memory storage space. The background storage is called the product data base. This fulfills the function of a long-term storage in which all data for a project may be accessibly stored for as long as desired. So the project data base may also be blocked against changes. The project data base is called passive when only reading access is allowed and active when all types of access may be performed.

The management of a project data base rests first on an arrangement of the data base in so-called pages. By this one understands storage regions with a constant number of storage spaces of FORTRAN type integer -- the so-called page size. The maximum size of the project data base, i.e. its largest possible page number, is a set property of the data base. This can span data bases of several operating systems. These must however be physically placed on storage devices which allow direct access (e.g. magnetic disk but not magnetic tape).

The working memory is a continuous storage region of definable size in the central storage of the computer. It is e.g. realized by the FORTRAN statement

```
COMMON // LOT (10000)
```

with a length of 10,000 storage spaces of FORTRAN type integer. This storage is available however only during the activity of the programming system. The logical connection of the project data base with the working storage space results in the following basic demands on the data processing system:

1. Addition of new pages to the project data base. Existing pages no longer needed must be processed along with the holes arising thereby. Thus the project data base always has the smallest possible size. /94

2. Transmission of data from the project data base into the working memory by copying pages

3. Dynamic processing of the working memory. Each access to data in the project data base follows only over one indicator into the working storage. The indicator is only available for use for a limited time. An external influence on the location of data is not possible.

On a high level of the data processing certain data modules are depicted on the pages. All required information about the linking of all corresponding pages are parts of each data module. Furthermore they contain all other data about their own architecture. One therefore denotes such data modules as self-descriptive. They are an important concept for the organization of access paths to data. Each type of data module is managed by its own functional unit. This is part of the data processing system. Self-descriptive data modules receive a special significance through the distribution of computer-connected and distributed data processing. Here above all the information about the data type stored in every storage space is of decisive significance in order to undertake a type-correct conversion between different computer systems. The data-processing system supports only a few standardized modules which favors a simple and easy-to-follow processing of data. For the requirements of above all the core system, the so-called tablets were introduced in [Schrem 78a]. They serve the storage of the hypermatrices which appear by the method of finite elements. Further, sequences and tables are provided for the storage of control lists and model data.

In the use of the data modules for the model description, /95 the two following principles are heeded:

1. Principle of unity of the model, mesh tree, and project data base (compare Chapters 3.3 and 4.2).

   > The project data base contains all data for one and only one model.

   That means that with a change in the mesh tree and thus in all of the model properties included, another model and thus another project data base must be chosen, because the user determines at the very beginning of the model descriptionby means of the mesh tree the context structure of the project data base and the structure of all data modules for the module data. This static structure of the project data base contrasts with a free changeability of the content of the data module, which may be formulated in the following way:

2. Principle of the generalized storage cell

   > A new datum always writes over an old one.

   For this, the creation of the data modules is in programming strictly separated from the changing of its contents. Through this, following every user input, the current state of data described up to the present is in the project data base. The data base is thus largely safe from interrruption.

It is expressly stated that these principles go back to design decisions for the programming system and are not required by limitations of the data-processing system. Therein a very fundamental design principle of the modularization of complex systems proves

itself useful. After that all lower abstraction levels of a system are conceived as generally as possible and the limiting design decisions follow first at higher abstraction levels. The freedom for such decisions is a direct consequence of the generality of lower abstraction levels. For the software system for the model description the principles named yield a simple implementation and an easy-to-follow user interface.

/96

5.2 Control Lists

5.2.1 Types

A programming system for modelling clearly has the task to offer the user a broad selection of various element types, so that he may describe his model in the proper form. The integration of this variety into the system can be achieved by a generalizer, a generalizing principle, as considered in the concept of the type. Each class of objects of the model description is described by a series of properties. Each type within the class is then characterized by a complete set of values for these properties. All elements are described e.g. by the number of nodes, but for each element type this number is set, and two different element types can be so differentiated. In general two types of a class differ in the values for at least one constitutive property. Further classes of objects are e.g.

- o meshes,
- o coordinate systems,
- o degrees of freedom, etc.

Now the principle of giving names and calling by name will be agreed upon:

> All types of each class of model objects receive an unambiguous type-specific name which is used by the user for identification of the type in the model description.

In addition, each type receives an alphanumeric type name from one to twelve characters. Such names are not arbitrary, rather they have for the user a mnemonic significance. For the processing of the types within the program system, however, numerical identifiers are better suited. Therefore each type name corresponds to an unambiguous type number. This is a positive integer greater than zero. From the point of view of programming techinique, the preference for numbers lies in lower memory demand, easier comparison, and the possibility to use them as indicators or keys, or to make calculations with them. /97

The properties of model objects of a class are now divided into parameterizable and non-parameterizable properties. Prerequisite for parameterizability is the possibility to specify definite numerical quantities for a property. Thus e.g. the quantity and types of degrees of freedom for an element are parameterizable properties of the element type. By contrast, the stiffness of an element is dependent on the configuration and must be newly calculated each time and so is not a parameterizable quantity of the element type. The differentiation concerned of the type properties is important for the strategy of the integration of many different types. Parameterizable properties can be written down on control lists. These fulfill the following assignments:

- naming.
- giving of the type name from the type number and vice versa.
- similar access to a property for all types of a class.
- easy increase in the number of types considered by the system.
- no burdening of the program text with any knowledge about properties of specific types.
- fixing of a canonical order of properties (e.g. the degree of freedom on a node of a mesh type).
- controlling data transformations (e.g. conversion of measurement units).

The control lists contain no mesh-dependent data and are therefore contained in the basic context of the project data base (compare Chapter 4.2).

For non-parameterizable properties of a specific type, programs must be integrated into the system. There the type number allows an unambiguous coordination of program and type in the form of a multiple branching (FORTRAN: computed GO TO). Thus, always with the help of a control list can be ascertained the number of an element type which makes directly possible the calling of the element program, e.g. for the stiffness calculation.

5.2.2 Sequences /98

For storage of control lists in the project data base, data modules of the type sequence are provided for. They obey the following rules of construction:

1. A sequence is a series of lists.
2. A list is a series of elements.
3. All elements of a list are atoms of the same type.

Each sequence is identified by a name. This consists of one to four characters. For control lists, those include a $ sign followed by one to four alphanumerical characters (compare Chapter 4.2.1). Each list within a sequence is, as well, designated by a name. This includes one to twelve alphanumeric characters (compare Chapter 4.2.1). Each list within a sequence is, as well, designated by a name. This includes one to four alphanumeric characters. thus these names are suitable for type names. In general the arrangement applies that:

- All type names are list names.
- All types of a class are summarized into a sequence.

Each element of a list is unambiguously determined by the specification of sequence name, list name, and index. The index has here the function of a number for designation of an element. As atom types, storage spaces of FORTRAN type integer or real come into consideration for control lists.

For the processing of sequences a functional unit is created which is a part of the data processing system. The state of each sequence is stored completely in the project data base. To this state belong:

1. The sequence is known/unknown

   With the function

   NEWSEQ (namseq)

   a sequence is created whose contents are empty. The quantity of possible entries is fundamentally limited. Moreover, there follows an entry into the contents of the active context. With this, the sequence is known. The /99 corresponding inverse function:

   DELSEQ (namseq)

   deletes a known sequence in the active context. All occupied pages in the project data base are freed up. The entry in the contents of the active context is removed. After this the sequence becomes unknown.

2. A list is known/unknown

   Each new list is first created by the function

   LISNEW (namseq, namlis[3], length, atom type)

The sequence concerned must be accessible in the active context.

A list with the given name may not already exist in the sequence. The function causes an entry into the list contents of the sequence. This corresponds to a change of state

numlis <--- numlis + 1 ,

where 'numlis' signifies the quantity of the known lists in the sequence. Moreover, the function causes the necessary storage spaces in the project data base to be reserved and initialized (if 'length" > 0). All elements already exist after that. Up to the calling of the function, a list has the state 'unknown'; afterwards, the state is 'known'. There exists for the following reasons no corresponding inverse function:

- o With an inverse function the coordination of list numbers and list names in the contents would be lost.

- o The management of the gaps thus arising in the contents and the sequence would be complicated and inefficient.

- o The use of sequences for storage of control lists makes the inverse function unnecessary, since control lists are only created a single time, but in contrast are used very often.

/100

For the storage of control information in a list, the following function is used:

3. Describing a list
   If a list is unknown (and 'length' > 0), then the contents of the list may be changed with the function

   LISWRT (namseq,namlis[3],istart,ifeld[nelem],nelem)

   This has write-over effects for the chosen index interval

   [istart,istart+nelem-1]

   From the set reservation of the list places, a lengthening of a list is not possible:

   $1 \leq istart \leq istart+nelem \leq length$.

The inverse function is

4. Reading a list

   LISRD (namseq,namlis[3],istart,ifeld[nelem],nelem)

   In reading and writing attention must be given to the atom type in order to make possible correct processing.

The functional unit encompasses a series of interrogatory functions:

5. State of a sequence

   INFSEQ (namseq,iex,numlis)

   The state variables are

   'iex' : The sequence is known in the actual context (iex=1) or unknown (iex=0). A change in the state variables is possible with DELSEQ or NEWSEQ.

   'numlis' : The number of the calls performed to LISNEW since the creation of the sequence.

THIS PAGE INTENTIONALLY LEFT BLANK

(Foreign page 101 missing)

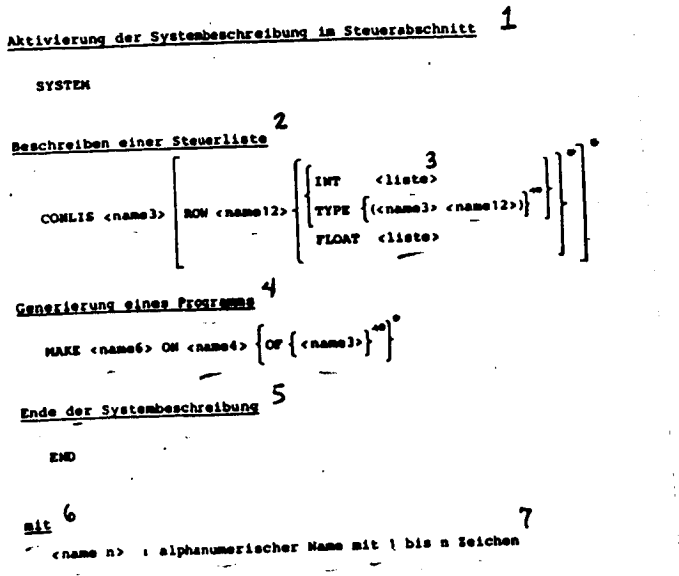
THIS PAGE INTENTIONALLY LEFT BLANK

(Foreign pg. 102 missing)

- o number of edges and their corner nodes /103
- o number of solid, plane, and edge middle nodes

2. Element configuration
   - o type of coordinate system
   - o number of linear interpolatable element node locations, the interpolation nodes, the support nodes, and the location parameters.
3. Element degree of freedom
   - o total number
   - o largest number at a node
   - o enumeration of degrees of freedom at each node
4. Element data
   - o number of possible model types
   - o for every model type: quantity, atom type, and dimensions of the element data.
5. Element loads
   - o number of possible load types
   - o for every load type: quantity, atom type, and dimensions of the load data

The control list may be tied into the system in two ways. One way is for the user to use a copy of the project data base containing all control lists. One speaks of a prepared project data base. The second way consists of an automatic creation of a FORTRAN subprogram which can accomplish the initialization and preparation of a project data base. In this way it is possible, following the mesh tree description, to include only the control lists in the project data base correlated with the mesh and element types used. If the data modules in the project data base are structurally self-descriptive (Chapter 5.1), then from the incorporation of the control lists, the project data base will also be self-descriptive with regard to the contents, since with their help the depiction of type names by numbers may be reversed at any time. The commands for description of the control lists and generation of FORTRAN subprograms are summarized in Fig. 5.2.



1 - Activation of the system description in the control segment, 2 - description of a control list, 3 - list, 4 - generation of a program, 5 - end of the system description, 6 - with, 7 - alphanumeric name with 1 to 5 symbols.

Fig. 5.2: Commands for system description.

5.3 Model Data

5.3.1 Numbering

The principle of naming and calling by name (Chapter 5.2.1) is now broadened as follows:

> All objects of the model description receive explicitly an unambiguous name which is used for identification of the object.

While names for all types are fixed by the system description, the following model objects are in addition arranged by the user:

- meshes,
- nodes,

- o elements, and
- o loading cases.

/105

The naming of these objects leads, in contrast to any implicit arrangement, to the user's exact knowledge of his model. Whether numbers or alphanumeric names are used depends on the number of objects and their use. Since commonly the number of nodes and elements in a model is large, alphanumeric names are little suited to the purpose. They would be extremely difficult for the user to memorize. Besides, numbers offer the advantage that they may be generated, if a regular number series is available (compare Chapter 6.5). Depending on the model, there could exist many or few meshes and loading cases. Their designation with alphanumeric names would be justifiable, but numbers will also be provided here for reasons of uniformity.

		1 Sequenz	2 Tabelle	3 Tablet	4 Block-tablet
5	Netzbaum	x	(x)		
6	Knotennummern		x		
7	Elementnummern		x		
8	Knoteninzidenzen		x		
9	Elementinzidenzen		x		
10	Inzidenzgrade		x		
11	Knotenkoordinaten		x		
12	Elementkoordinaten		x		
13	Knotenbasen		x		
14	Elementbasen		x		
15	Netzbasen	(x)	x		
16	lokale Basen	(x)	x		
17	FG-Klassen-Zuordnung		x		
18	FG-Klassenselektionsmatrix			x	
19	FG-Inzidenzmatrix			x	
20	Elementdaten				x
21	Lastfälle	(x)	x		
22	Elementlasten				x
23	Knotenlasten			x	
24	vorgeschr. Verschiebungen			x	
25	Elementsteifigkeit			x	
26	Elementkräfte			x	

1 - sequence, 2 - table, 3 - tablet, 4 - block tablet, 5 - mesh tree, 6- node numbers, 7 - element numbers, 8 - node incidences, 9 - element incidences, 10 - degrees of incidences, 11 - node coordinates, 12 - element coordinates, 13 - node bases, 14 - element bases, 15 - mesh bases, 16 - local bases, 17 - class of degree of freedom, 18 - degree of freedom selection matrix, 19 - degree of freedom incidence matrix, 20 - element data, 21 - loading cases, 22 - element loads, 23 - node loads, 24 - given displacements, 25 - element stiffness, 26 - element forces

Fig. 5.3: The model data and their data modules

Out of the order of the naming an ordinal numbering results for all objects. These internal numbers are used for data processing. In contrast to this the numbers given by the user are designated external numbers. The correspondence of external to internal numbers is a property of th object in question and belongs to the model data. The coordination of all objective data results from the user by the external number of the object and by the project data base from the depiction by the internal number.

Which type of data module is chosen for the storage of the model data depends on the expected operations. Fig. 5.3 shows the summary of all model data and the corresponding type of data module chosen for each. If matrix operations are expected, then tablets are best suited for the storage of the data. Block tablets are used for the storage of data which is used in the element calculations. Reference is made to [Schrem 78a], in which can be found a thorough depiction of the tablet concept. For a series of model data, the direct access to a data bundle by an index (the internal number) is desired. Then the tablets described below are suited to the task. In some cases sequences are alternatively possible if an alphanumeric name is available for the access (e.g. the context name for meshes in the mesh list).

5.3.2 Tables

The rules of construction for tables are:

1. Each table is a series of lines
2. All lines possess the same number of columns
3. All columns consist of one atom of each of the same type.

The name table for this type of data module is derived from the identification of each column with a name which corresponds to a list name in sequences. Besides this the access always occurs on a /107

whole line of the table. For this the lines are implicitly numbered ordinally. The tables allow e.g. a relation between the internal node numbers and the degrees of freedom whose mesh-type-dependent names stand over the columns. If the number of atoms in each line is n, then each index in the table corresponds to an n-tuple of values. The significant difference with sequences is that the lines of the sequences correspond to the columns in the table. Despite this the atoms in each line occupy consecutive memory spaces in both types of data modules. Such an arrangement is decisive for an efficient access to the data. The dimensions of each table are determined on one hand by the control lists and on the other hand by the mesh tree. If both are known, the structure of the table is fixed: the names of the mesh-type-dependent degrees of freedom are e.g. taken from one control list and placed over the columns, while the length of the table is determined by the number of nodes of the mesh tree.

For the management of the tables a functional unit in the framework of the data processing system is available for use. As required by the similarity to sequences, the functions are also arranged in a similar manner (compare the flow charts in Figure 5.1 and 5.4). The state of the tables is changed by the following functions:

1. The table is known/unknown
   A table is made known and created by the function

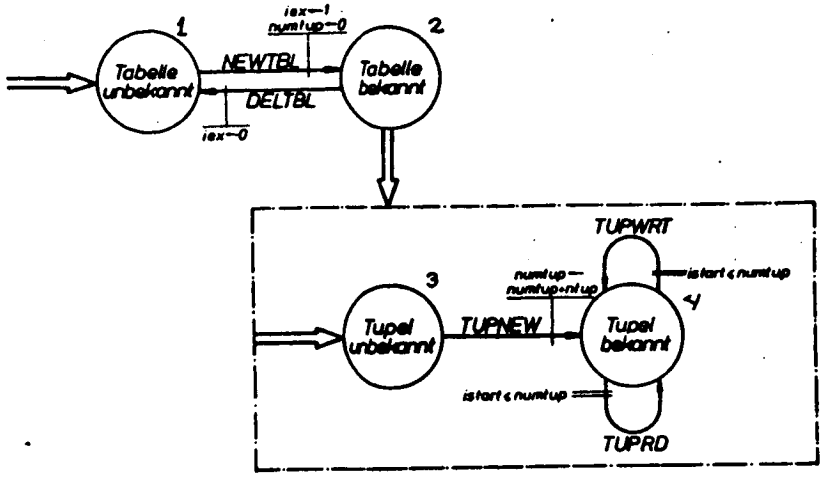
   NEWTBL (namtbl, atomtyp, numcol, namcol[3, numcol]

   The name of a table may here contain one to four alphanumeric symbols. The table possesses from the beginning a fixed column directory, which belongs unchangeably to the state of the table until the inverse function

   DELTBL (namtbl)

   is called. Both functions refer to the same context. Like a list name of a sequence, a column name also has one to twelve alphanumeric symbols.

/108



1 - table unknown, 2 - table known, 3 - tuple unknown, 4 - tuple known

Fig. 5.4: Flow chart for the management of tables.

2. The tuples are known/unknown
Each tuple is unknown until it is added to the table by the function

TUPNEW (namtbl, ntup, istart)

By this function are created the next 'ntup' tuples, starting with the tuple 'start' (normally 1). The change in state of the table is

numtup <-- numtup + ntup

where 'numtup' stands for the number of the available tuples in the table. Besides this the function causes the necessary storage space in the project data base to be reserved and initialized. A corresponding inverse function is not provided for, for reasons analogous to the sequences.

/109

The following function serves for storage of model data:

3. Description of one or several tuples

TUPWRT (namtnl, numcol, istart, ifeld[numcol, ntup], ntup)

Beginning with the tuple on the index 'istart', tuples are described by the contents of the field 'ifeld'. Then all tuples must be created (i.e. with the function TUPNEW):

$$1 \leq \text{istart} \leq \text{istart} + \text{ntup} - 1 \leq \text{numtup}$$

The corresponding inverse function is:

4. Read one or more tuples

   TUPRD (namtbl, numcol, istart, ifeld[numcol, ntup],ntup)

In addition to the functions named, the functional unit includes a series of interrogatory functions:

5. State of a table

   INFTBL (namtbl, iex, atomtyp, numcol, numtyp)

   The state variables are:

'iex'	: The table is known in actual context ('iex'=1) or unknown ('iex'=0). A change in the state can occur by DELTBL or NEWTBL.
'atomtyp'	: type of atom of which the entire table consists (commonly atoms are FORTRAN type 'integer' or 'real'
'numcol'	: number of columns in the table
'numtup'	: number of tuples created with TUPNEW since the origin of the table

6. Inquiry of column names

   For one thing, the column name may be determined for a given column number using

   TUPCOL (namtbl, icol, namcol[3]) ,

   for another thing the inverse

   TUPIND (namtbl, icol, namcol[3], icol)

   determines the column number for a given column name. While in the first function the column must exist (compare INFTBL), by the second function the column number is set to zero if in the table no column of this name is available.

If the content of a tuple is known, but the index in the table is desired, the function

7. Searching for a tuple

   TUPSRC (namtbl, istart, numcol, ifeld[numcol], index)

helps. Beginning with the tuple on the index 'istart' the first tuple is seached whose contents agree with that of 'ifeld'. Its index is determined. It is zero when no such tuple is found by the end of the table.

5.3.3 Project Maintenance

The principle of unity of the model, mesh tree, and project data base ensures a simple implementation of the model description. This is based above all on the separation of the following tasks from the model description:

- o the combination of partial modules
- o the broadening or shrinking of the mesh tree /111
- o changing the number of nodes, elements, or loading cases
- o the use of other element types, model types or load types.

The processing of all these tasks always leads to a new model and thus to a new project data base. For the user, a tool is now made available which allows him to transfer parts of already-existing models into the new model in order to make unnecessary a repetition of parts of the model description. In addition a further segment-like functional unit is created which does the following:

- o creation and initializing a new project data base.
- o copying certain data modules from an old to a new project data base.

Old and new project data bases are both managed by the data processing system. Therefore the page sizes concerned must agree (compare Chapter 5.1). Before copying, a new project data base must -- following the abovementioned principle -- first be described with a new mesh tree. Here the new mesh numbers may not be chosen with complete freedom. In particular, they may not be identical to the old mesh numbers. Through this, an unambiguous coordination of the mesh

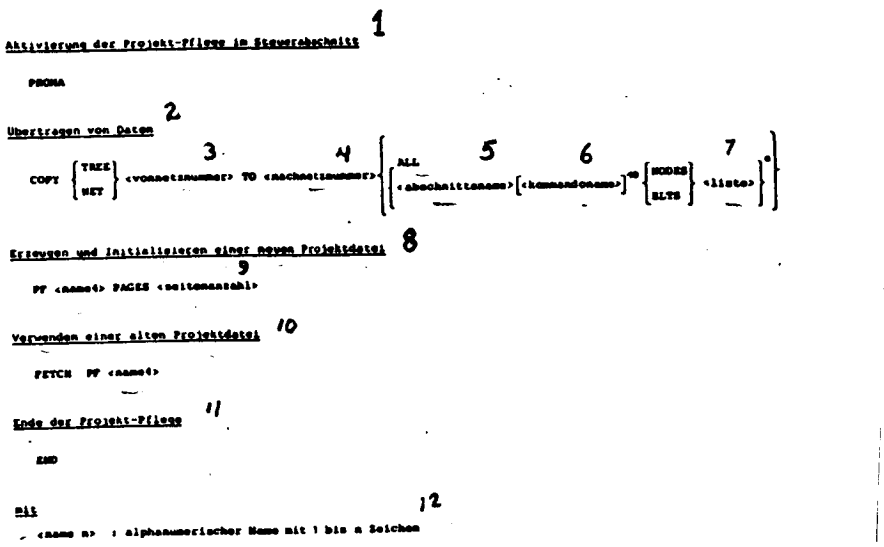
contexts is possible for the data processing system. The new mesh tree must be consolidated. A further condition for the application of the project maintenance package is given by the strict separation of creation and changing of data modules (compare Chapter 4.1). All the segments of the new project must be previously initialized with a first call, for which certain data modules are transferred from old project data bases (compare Fig. 4.1). If all these conditions are fulfilled, the user can call the following copy routines:

1. copying from a mesh or a lower tree
2. copying all data modules of the model description or such out of certain segments or for certain descriptive commands
3. Copying data for certain nodes or elements

/112

For all copy processes the compatibility of the data modules concerned is required.

After the application of the project care package the user can complete his model description in the manner already known. An especially important point here is, that all segments for the new project must be consolidated (according to the principle of consolidation, Chapter 4.1.1), in order to guarantee the consistency of copied data modules. For old project data bases this demand need not be fulfilled as long as no calculation should take place for the model concerned. With the help of the project maintenance, the user can for one thing access back to already-existing models (project library); for another thing, it is possible to modularize complex calculation tasks into partial models with their subsequent assembly. The user commands necessary for this are summarized in Fig. 5.5



1 - Activation of project maintenance in the control segment, 2 - transmission of data, 3 - mesh number copied from, 4 - mesh number copied to, 5 - segment name, 6 - command name, 7 - list, 8 - creation and initialization of a new project data base, 9 - number of pages, 10 - use of an old project data base, 11 - end of project maintenance, 12 - with <namen> : alphanumeric name with 1 to n symbols.

Fig. 5.5 : Commands for project maintenance

6. Command Language

6.1 Overview

The application of a satellite system for model description brings a division of work favorable to the user. While he idealizes and describes a model, the model data are processed and managed by the system. The form of the model description determines significantly the scope of the work reduction for the user and thus the user friendliness of the system. As a basis for the communication of the user with the satellite system, very different processes may be used such as command languages, menu, and dialog processes. The choice of a command language is based on its general applicability. They can be used in the same way on a CRT-device as well as on a line printing terminal. Besides this it is suited for self-documentation of the communication. A series of commands can be easily processed again and again without the user doing the same work each time. In the foreground, the command language serves the communication of the user with the system. But a more important aspect lies in the communication among different users. A series of commands describes intentions and processes for the system and every other user in a form understandable to all.

In a definite contrast to the demands of the users on the form of the model description are the needs of the programming system for an efficient supply with model data. On one hand, the user wants the largest possible freedom in the organization of the model description (keyword: free format), on the other hand, the processing of the model data is made much easier by a standardization of data depiction (keyword: fixed format). The resolution of this conflict lies in the creation of a logical interface between user and programming system which converts from free to fixed format.

In the establishment of the user interface, above all, three sources are drawn from. First, problem-oriented languages (POL's) are

to be mentioned. This concept was developed in the early days of /114
the application of computers, as it was desired to make the possibilities of computers accessible, without requiring mastery of a programming language (compare [Fenves 64, Roos 66]). These languages are of a simple type and their sentences frequently connect certain portions of data with memorable keywords taken from the jargon of the particular area of application. Thus, the user is in a position to describe his problem and determine its processing without giving the underlying algorithms. There lies the significant difference between these languages and algorithm-related programming languages (procedure-oriented languages). While a programming language serves the implementation of a programming system, a problem-oriented language takes care of the communication of the users with the system. The second source is represented by operating system control languages. They serve the user of a computing system for the description of a desired process. To this belongs the demands of operating equipment and the output control. These are often very simply constructed, that is, the commands consist only of a keyowrd followed by a -- usually short -- parameter list. It is characteristic of many command languages that there are several command planes to differentiate. Among these one understands a state of the command interpreter, which can be changed by certain commands such that afterwards other commands apply as before (e.g. calling and exiting from a text editor). Since each command plane has its own commands and the operations belonging to these, the easy change of planes signifies an integrating factor for several partial systems of an operating system. It is desired as well that new commands and command planes may be integrated into an existing system without great difficulty. Besides this, the possibility to be able to establish command procedures and the suitability of the command structure for interpretation and compilation belong to the demands on a modern command language [Gram 75].

The third source is from the theory of formal languages (e.g. [Aho 72, Salomaa 73]) just as it also forms a basis for the

development of programming languages. Out of this, exclusively /115 regular languages (Chomsky Type 3 [Chomsky 56]) come into consideration, whose construction is simple and sufficient for the model description. The choice of this language class limits in a useful way the number and complexity of rules of language creation and language recognition, the so-called syntax.

With the help of the source named, the following strongly simplified structural description of the interface is possible. The component command language allows the user to describe his model and to control the course of the model description. Each segment receives its own so-called segment language as a building block of the satellite system. The control segment represents the highest command level, from which the several equally-ranked segments of the other segments are called. Corresponding to this vertical language structure is a horizontal language structure in the form of segment-specific and general commands. The latter form a basic language common to all segment languages. The segment-specific commands are carriers of the model description in the sense of problem-oriented languages, while the general commands are used for communication control in the sense of the operating system control languages. As aspects of user-friendliness, the following minimum demans are to be placed on the languages to be developed:

- o Ease in learning: The user should, after a short introduction, be in a position to describe a model by himself.
- o Good readability: The user should be able to arrange the command input comprehensibly. A sufficient use of type names and key words ensures good readability.
- o Simple applicablity: The user should not have to pay attention to complicated punctuation rules. Meaningful key words alone identify the data unambiguously.
- o Little writing: The user should be able to describe as much data as possible with as few statements as possible.

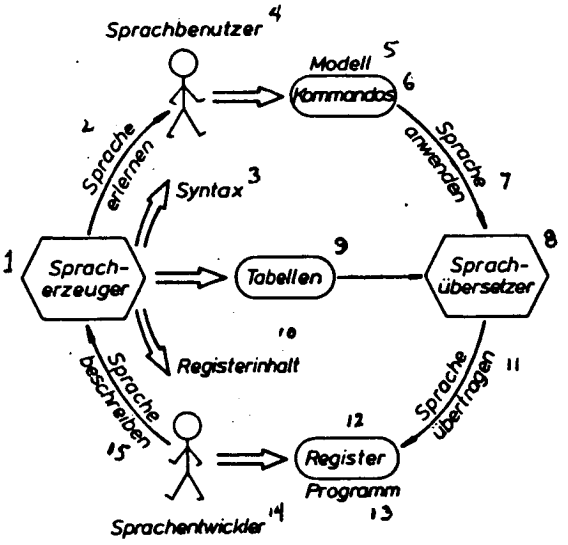
The so-called register may be named as the second component of the interface on the program side. This name is chosen on the register management during compiliation of programming languages. As there, a defined datum exists during a definite time span through which its efficient processing is guaranteed. Following the input of a command, the register considered here receives the command's entire data content in a precisely prescribed form. The content of the register stays unchanged until the demand of the next command. A register consists of three parts:

1. Identifier part: this determines the type of command depicted in the register.
2. Descriptor part: this describes the content of the following parts of data according to type and availability of certain data proportions.
3. Data part: this contains all data packed in the command in a command-specific order.

The length of the register and its parts are fixed for a language. Through this, since also the location of certain data in the register is fixed for every command, it is possible for the processing program to access directly each datum.

The core of the sought-for interface is formed by the dual component pair of language creator and language translator. Their roles should be explained by Fig. 6.1. First, the developer determines the command language of a new segment, with which the user will input the segment-specific model data. This language is then described by means of the so-called definition language. This description is processed by the language creator out of which results a three-fold result. For the future user, an assembly is created with the syntax of the command language. The developer of the segment receives a description of the contents of the register belonging to every command of the language. With this information, the programming

of the processing of the model data may be very easily arranged. The third result is represented by a set of tables which make it possible for the language translator to recognize commands of the language as such and to implement their transmission into the register. In this type of procedure result some important consequences: /117



1 - language creator, 2 - learn the language, 3 - syntax, 4 - language user, 5 - model, 6 - commands, 7 - apply the language, 8 - language translator, 9 - tables, 10 - register contents, 11 - transmit the language, 12 - register, 13 - program, 14 - language developer, 15 - describe the language.

Fig. 6.1: The command language forms the user interface.

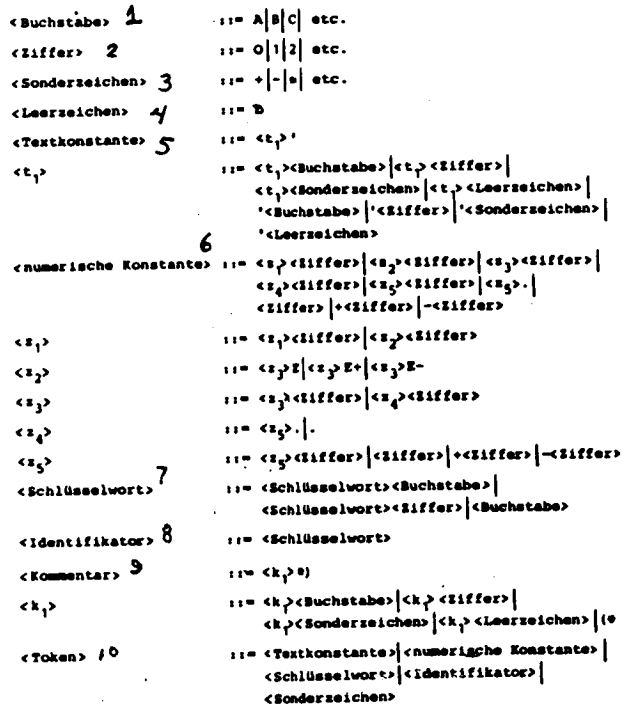
- There is only one language translator. It understands in each case the language whose tables are made known to it. The language is table-driven.
- There is exactly one set of tables which the language creator makes from the description of the definition language and with which the language translator can understand the definition language. Thus the definition language is self-descriptive and complete.
- Only such language construction is possible as may be described with the definition language. This is language-limiting because through its descriptive possibilities a limited class of languages is defined.
- The language translator possesses no type of knowledge about the meaning of the contents of commands. It is exclusively syntax-oriented and therefore free of semantics.

Thus is outlined the interface,which lets the user see only the front side of the satellite system. Each command input by him is transmitted in the first phase from the language translator in register form. This depicts in a certain way the workshop façade of the system. Behind it follows in the second phase the interpretation of the contents of the register. These two steps correspond to the division of command processing according to the viewpoints of syntax and semantics. Through this division, the language translator creates a general as well as a simple tool for the processing of the model description.

6.2 Lexical Analysis and Free Format

The answer to the question whether a command depicts a correct sentence of the presently active language is the prerquisite for the transmission of the command into register form. This is the task of the so-called syntactic analysis. Extensive experience is present on this subject primarily from the compilation of programming languages (e.g. [Gries 71, Aho 72, Bauer 76]). Here an important role is played by the determination of the words of a sentence. This happens by means of lexical analysis: From every series of signs is filtered out that string of symbols which represent the smallest units (words) of the language, the so-called tokens. They are based on a set of symbols of the alphabet out of which they come forth through definite rules of construction.

An explanation of the lexical analysis suited for the needs of the command language is documented in [Schrem 79]. By this, each token is formed as a regular expression out of an alphabet of standardized symbols. This may be shown with the aid of the left-linear production system shown in Fig. 6.2 (for proof, see e.g. [Salomaa 73]). In the following, some concepts of this lexical analysis will be named, in order to be able to judge that aspect of the user-friendliness known under the keyword "free format".



1- letter, 2 - number, 3 - special symbol, 4 - spacer symbol, 5 - text constant, 6 - numerical constant, 7 - keyword, 8 - identifier, 9 - comment, 10 - token

Fig. 6.2: Left-linear production system for the tokens of the command language.

The input of the command language occurs line-by-line, by which each line has a fixed length of 80 symbols. The end of a line is reached either after 80 symbols ('physical end-of-line') or after the symbol 'logical end-of-line' ('*)' at the beginning of the line '␣␣ ' on an entire line). Five token types are to be differentiated:

1. text constant,
2. numerical constant,
3. keyword,'

4. identifier,
5. special symbol.

/120

The token types numerical constant, keyword, and identifier must be separated from one another by at least one spacer symbol, one special symbol, or one end-of-line symbol. Therefore, within tokens of these types neither spacer symbols nor unallowed special symbols may appear. Besides this, all tokens must lie completely within one line.

In view of these conventions, the input for the user is format-free for the following reasons:

- o Tokens can stand anywhere between the beginning and the physical or logical end of a line.
- o Everywhere in the input (outside of tokens) comments may be added. They may be added either between opening and closing comment parens (symbol (* and *), repectively) and can extend for many lines or between logical and physical end-of-line.
- o Whether a numerical constant represents an integer depends not on the depiction of the number but rather on its value (e.g. 0.5E1 and 5 are different depictions of the integer 5). Therefore the user must not be concerned with the differentiation of real numbers and integers known from FORTRAN in the input of a number.

The advantage of the format-free-input for the user is that he can deal with the ordering of the language elements in free form as it appears correct for him. From this results however a responsibility of the user for the comprehensibility of the chosen order, if his colleagues are supposed to understand the input text. The given definition of format-free input is very general. Commonly, this property is only connected with the free distribution of numerical values in an input line as opposed to the fixed format prescibed, for example, in FORTRAN [DIN 66027]. The extension to different token types leads to a more comprehensive meaning of the term.

Noteworthy is the fact that the format-free input represents exclusively a property of the word formation and not the sentence formation of the command language. This means the introduction of an important abstraction step through the lexical analysis. Only the order of the words themselves, not their distribution on the input lines is the object of the syntax of the command language.

6.3 Syntax

If one considers the lines processed by the lexical analysis as the physical sentences of the language, the commands then represent the logical sentences of the language. For them now a syntax should be declared which follows the fundamental conditions of the command language for the model description:

- o right for the user through a simple (because regular) language
- o right for the application through data-descriptive, non-algorithmic language (keywords are connected with certain portions of data).

All languages that satisfy the following rules form a class of languages. Therefore no terminal symbols, rather only non-terminal symbols of the language are required [Aho 72]:

1. Each command language consists of a finite number of different commands.
2. Each command consists of a series of parameters.
3. Each parameter consists of a (parameter-) keyword and a subsequent, possibly empty parameter list.
4. All parameters obey a fixed predecessor/successor relation, by which each parameter is identified by its predecessor and its keyword and even the quantity of possible successors is determined. By this relation, sentence symbols become unnecessary.
5. Some parameters are differentiated on the basis of /122 their syntactic function within a command:

- o Beginning parameter: This is the first parameter of a command. Its keyword is called the command keyword and identifies the entire command. There is exactly one beginning parameter for each command.
- o Loop parameter: By repetitions of the same commands immediately following one another, the first parameters can be identical, so that it suffices to intorduce these parameters in only the first command of the series. For this, certain parameters in the command take the function of loop parameters so that the command part preceding need not also be repeated by repetition of the command. A beginning parameter cannot be a loop parameter at the same time.
- o End parameter: Such a parameter is denoted by the quality that it is followed by either a loop parameter of the same command or the beginning parameter of the next command. An end parameter can also be a loop parameter or a beginning parameter, but not both at the same time.
- o Optional parameter: These are the parameters of a command which need not be given in every case when the command is formulated. Their choice is influenced by semantic aspects. Each parameter of a command besides the beginning parameter may be an optional parameter.

6. In parameter lists one differentiates two forms

   - o Closed parameter lists: they possess fixed minimum and maximum lengths l (that is, the number of values in the lists):

     $$0 \leq l_{min} \leq l \leq l_{max}$$

     As the values, all token types come into consideration. For each position in the list the lower limit of allowed

token types as well as the value range of the token of each type (token sub-type [Schrem 79]) must be fixed.

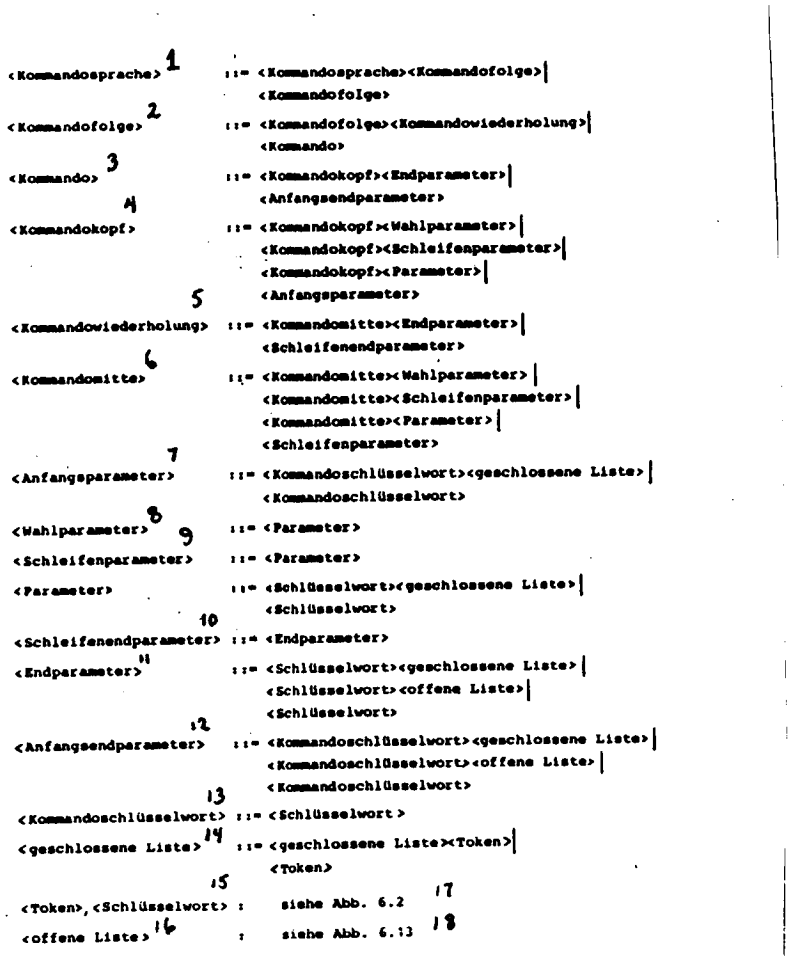
- o Open parameter lists: Their length has no upper limit: /123

$$l \geq 1$$

As values are allowed exclusively tokens of the type numerical constant. Through this limitation the list is imaginable as a series of numbers. Thus it is suited for the depiction of the most model data, whose quantity depends on the model. /124

7. A branching in a command occurs when many parameters of the command have the same predecessor. A union signifies that a parameter of the command has several predecessors. Thus it is possible to bundle parameters which belong together into one command.

The syntax described now is depicted in Fig. 6.3 as a production system. Here the predecessor-successor relation is not considered, because to describe them, the terminal symbols of the language are needed. As already mentioned, the introduction of sentence symbols is made unnecessary by this relation. The beginning and end of a command are determined unambiguously by the corresponding parameters. The input of commands can thus occur free of punctuation. To be sure, many punctuation marks are provided for (as , and ;) but their use for arrangement of the input is according to the preference of the user. There remains one more important consequence to be drawn from properties of format-free and punctuation-free input: If a logical sentence is distributed over several lines, then no continuation characters (as in FORTRAN [DIN 66027]) are necessary. This proves to be user-friendly, especially in the input of commands using a CRT-terminal.



1 - command language, 2 - command order, 3 - command, 4 - command level, 5 - command repetition, 6 - command middle, 7 - beginning parameter, 8 - optional parameter, 9 - loop parameter, 10 - loop end parameter, 11 - terminal parameter, 12 - beginning terminal parameter, 13 - command keyword, 14 - closed list, 15 - keyword, 16 - open list, 17 - see Fig. 6.2, 18 - see Fig. 6.13

Fig. 6.3: Production system for the class of command languages.

6.4 Aspects of Language Processing

6.4.1 Language Recognition

The determination, whether or not a sentence with a definite vocabulary is a sentence of a language already presented, belongs to the task of language recognition or syntactic analysis. For regular languages the deterministic finite automat (DFA) represents a mathematical model of the syntactic analysis [Böhling 69, Aho 72]. For the recognizing automat a representation is chosen in the form of a language matrix. Its columns represent the vocabulary V* of a command language. This encompasses the quantity of the parameter keywords and a blank word V_e. The rows of the matrix represent /125 the number of thestates of the automat. Each state is understood by exactly one correct word out of the vocabulary of the language, that is, each word V_i corresponds to exactly one state S_i. The present state of the automat during the recognition process is therefore given unambiguously by the line index in the language matrix. The language matrix is a Boolean matrix, in which each row and each column contain at least one 1. A change of state δ_{ij} is then the change of the row index i on one of the column indices j, in the current state of which, row i contains a 1. Or, put another way: each line i, which stands for an already known word V_i, contains for all successors V_j a 1 in column i and otherwise 0. The language matrix can therefore be seen as a depiction of the predecessor-successor relation of the parameters of the command language. The initial state S_n of the automat corresponds to the blank word V_e of the vocabulary and has all command keywords of a language as successors.

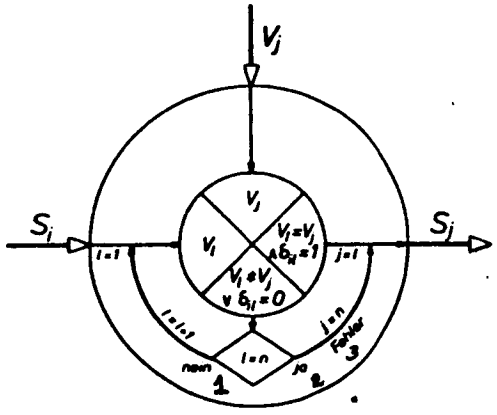
A word V_j is now recognized by the automat if it is in state S_i and is valid for the contents of the language matrix $\delta_{ij} = 1$. Thus one operator may be defined for all changes of state which, based on the next input word, shifts the momentary state of the automat into the next one. This recognition operator

$$\xi(\delta_{ij}) = \xi(s_i \xrightarrow{v_j} s_j)$$

is shown in Fig. 6.4. In the case of a mistaken input it performs a change of state to the initial state S_n. Its effect may be summarized as

$$s_i \xi v_j = \begin{cases} S_j, & \text{for } \delta_{ij} = 1 \\ S_n, & \text{for } \delta_{ij} = 0 \end{cases}$$

The way the operator works is to compare the state Si of all the values V_l for which $S_{il} = 1$ with the input word V_j. If the search is successful, the desired change of state will occur. In the other /126 case the automat reverts to the initial state. The input of the blank word V_e is recognized if $\delta_{ij} = 1$, which is always the case after terminal parameters.



1 - no, 2 - yes, 3 - error

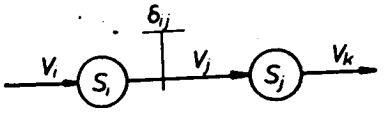
Fig. 6.4: The recognition operator $\xi(s_i \xrightarrow{v_j} s_j)$

6.4.2 Graphic Depiction of the Syntax

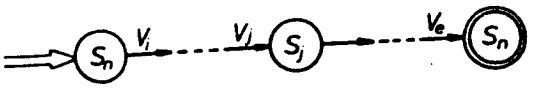
The language matrix is not suited for the depiction of a command language outside of the recognizing automat. In contrast, graphic depictions are above all suited to give a quick overview of a language. Since the order of the language is decisive for the syntax, directed graphs are used (e.g. Dörfler 72]).

Depending on which constitutive quantity of the command languages occupies the nodes of the graph, one can differentiate three equivalent depictions:

1. State Diagram: The states of the recognizing automats /127 form the nodes of the graph:

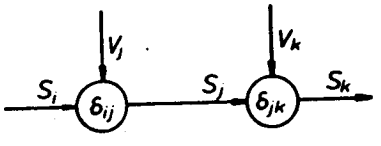


The relationship of this diagram with the language matrix is that this diagram depicts the adjacence matrix of the state diagram [Dörfler 72]. For each command the state diagram begins and ends in state S_n:

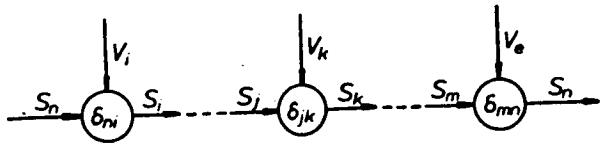


The graph has as many nodes as the command parameter has, plus the beginning and end nodes.

2. State transition diagram: Here the nodes are occupied by the state transitions:

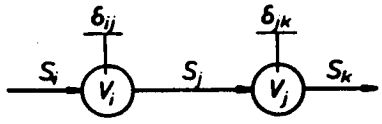


One can imagine such a diagram as arising from the subsequent switching of recognition operators ξ (δij) (compare Fig. 6.4). For a command language, it has as many beginning nodes as the language has commands and as many end nodes as there are available end parameters:

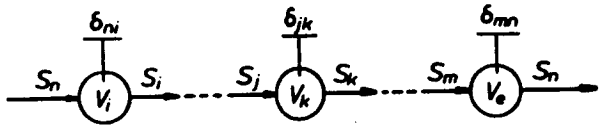


/128

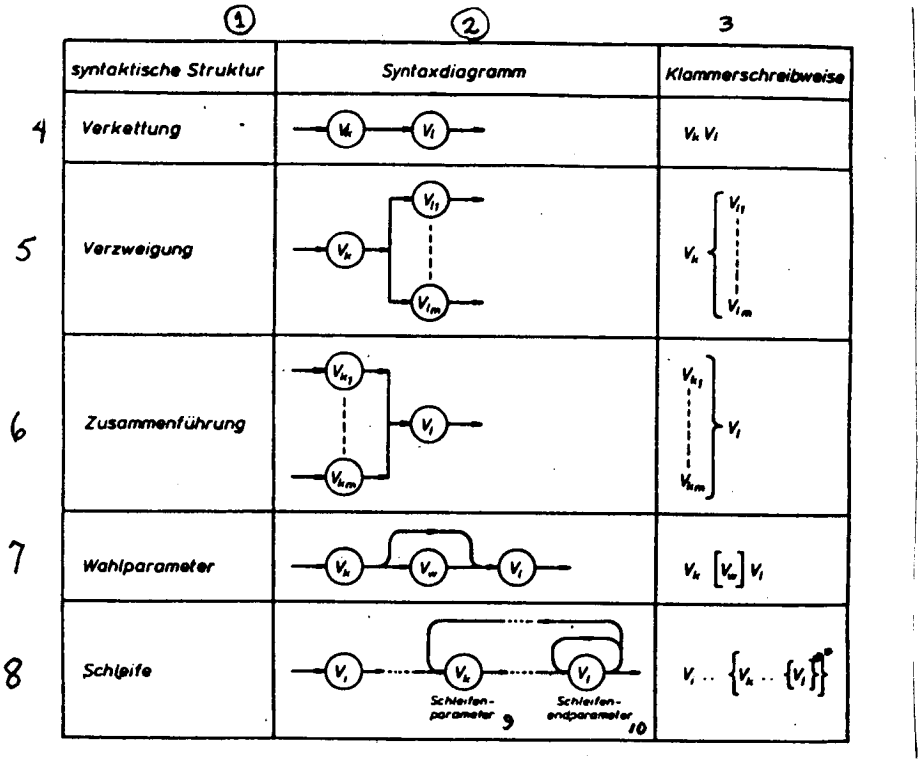
3. Word diagram: With this, the words of the language are depicted through the nodes:



Word diagramms of single command begin with the beginning parameter and end with the blank word:



Here the relationship between the graphical depiction of regular languages and functional units should be noted [Schrem 78b]. Use was already made of the descriptive possibilities of a functional unit through a diagram of state. A further analogy exists between syntax diagrams and calling diagrams (compare Figs. 6.11 and 6.12). The observed relationship is thus based on the regularity [Salomaa 73] of the appropriate rules of construction.



1 - syntactic structure, 2 - syntax diagram, 3 - method of writing with parentheses, 4 - linking, 5 - branching, 6 - merging, 7 - optional parameter, 8 - loop, 9 - loop parameter, 10 - loop end parameter.

Fig. 6.5: Two depictions of the syntactic structures of the command language

6.4.3 Language Description

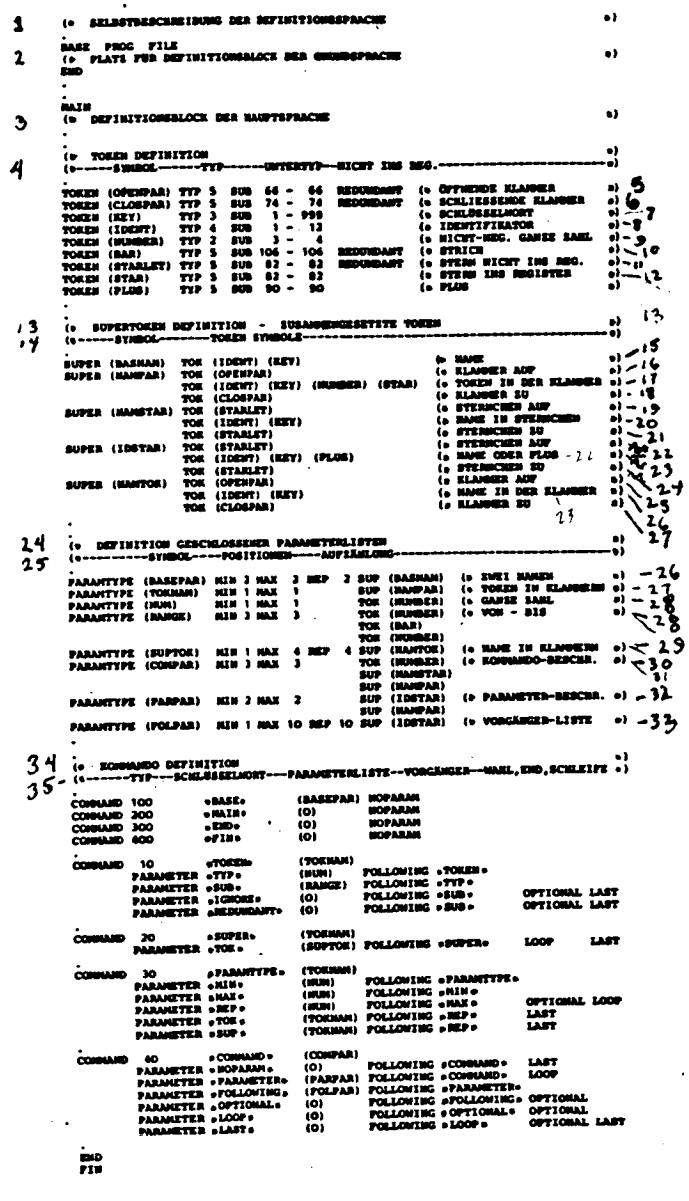
To be sure, the language matrix is suited for control of the syntactic analysis of a command language, since it has all permissible sentence forms, but it does not describe the language completely. It encompasses only commands and their parameters. The fine structure of parameters also needs to be arranged, and for that there must be suitable concepts found, which, as the language matrix, are applicable for the entire class of the command languages. The description /130 of a parameter includes the following information:

1. Parameter keyword
2. Parameter list type
3. Type of list (open/closed)
4. Description of closed lists
   a) minimum length (in positions)
   b) maximum length (in positions)
   c) number of tokens for every position
   d) for every token the possibility for the token type and its range of values (token sub-type)

This information is organized in the so-called parameter table by which the keywords and the position descriptions are comprehensible by reference to the keyword table or the token table, respectively. With this, there is a set of language tables, which contains all data necessary for a complete description of the language:

1. language matrix,
2. parameter table,
3. keyword table,
4. token table

The creation of the language tables can be automated with the help of a functional unit -- the language creator. Its input for this consists of an exact language description by the means of the so-called definition language. While the tabular form of a language is suited for the recognizing automat, the linguistic description has significant advantages for the developer of new commands. Among these may be named good readability and ease of changing the description. Since the definition language itself is an element of the language class considered, its qualities may be best explained by a self-description. Figure 6.6 shows the complete description of the definition language. According to the conscious separation of syntax and semantics, in the description are described exclusively syntactic units and sentence forms. The power of the language amounts to 24 parameters in 8 commands. These suffice to describe every command



```
1   (* SELBSTBESCHREIBUNG DER DEFINITIONSSPRACHE                          *)
.
2   BASE  PROG  FILE
    (* PLATZ FUER DEFINITIONSBLOCK DER GRUNDSPRACHE                      *)
    END
.
    MAIN
3   (* DEFINITIONSBLOCK DER HAUPTSPRACHE                                 *)
.
.
    (* TOKEN DEFINITION                                                  *)
4   (*-----SYMBOL-------TYP------UNTERTYP---NICHT INS REG.--------------*)
.
    TOKEN (OPENPAR) TYP 5  SUB  66 -  66  REDUNDANT  (* OFFNENDE KLAMMER      *)
    TOKEN (CLOSPAR) TYP 5  SUB  74 -  74  REDUNDANT  (* SCHLIESSENDE KLAMMER  *)
    TOKEN (KEY)     TYP 3  SUB   1 - 998             (* SCHLUESSELWORT        *)
    TOKEN (IDENT)   TYP 4  SUB   1 -  12             (* IDENTIFIKATOR         *)
    TOKEN (NUMBER)  TYP 2  SUB   3 -   4             (* NICHT-NEG. GANZE ZAHL *)
    TOKEN (BAR)     TYP 5  SUB 106 - 106  REDUNDANT  (* STRICH                *)
    TOKEN (STARLET) TYP 5  SUB  82 -  82  REDUNDANT  (* STERN NICHT INS REG.  *)
    TOKEN (STAR)    TYP 5  SUB  82 -  82             (* STERN INS REGISTER    *)
    TOKEN (PLUS)    TYP 5  SUB  90 -  90             (* PLUS                  *)
.
.
13  (* SUPERTOKEN DEFINITION - ZUSAMMENGESETZTE TOKEN                    *)
14  (*-----SYMBOL-------TOKEN SYMBOLE-----------------------------------*)
.
    SUPER (BASNAM) TOK (IDENT) (KEY)                    (* NAME              *)
    SUPER (NAMPAR) TOK (OPENPAR)                        (* KLAMMER AUF       *)
                   TOK (IDENT) (KEY) (NUMBER) (STAR)    (* TOKEN IN DER KLAMMER *)
                   TOK (CLOSPAR)                        (* KLAMMER ZU        *)
    SUPER (NAMSTAR) TOK (STARLET)                       (* STERNCHEN AUF     *)
                   TOK (IDENT) (KEY)                    (* NAME IN STERNCHEN *)
                   TOK (STARLET)                        (* STERNCHEN ZU      *)
    SUPER (IDSTAR) TOK (STARLET)                        (* STERNCHEN AUF     *)
                   TOK (IDENT) (KEY) (PLUS)             (* NAME ODER PLUS    *)
                   TOK (STARLET)                        (* STERNCHEN ZU      *)
    SUPER (HANTOK) TOK (OPENPAR)                        (* KLAMMER AUF       *)
                   TOK (IDENT) (KEY)                    (* NAME IN DER KLAMMER *)
                   TOK (CLOSPAR)                        (* KLAMMER ZU        *)
.
24  (* DEFINITION GESCHLOSSENER PARAMETERLISTEN                          *)
25  (*----------SYMBOL----POSITIONEN----ANFZAHL-ENDZ------------------*)
.
    PARAMTYPE (BASEPAR) MIN 2 MAX  2 REP  2 SUP (BASNAM)  (* ZWEI NAMEN       *)
    PARAMTYPE (TOKNAM)  MIN 1 MAX  1        SUP (NAMPAR)  (* TOKEN IN KLAMMERN *)
    PARAMTYPE (NUM)     MIN 1 MAX  1        TOK (NUMBER)  (* GANZE ZAHL       *)
    PARAMTYPE (RANGE)   MIN 3 MAX  3        TOK (NUMBER)  (* VON - BIS        *)
                                            TOK (BAR)
                                            TOK (NUMBER)
    PARAMTYPE (SUPTOK)  MIN 1 MAX  4 REP  4 SUP (HANTOK)  (* NAME IN KLAMMERN *)
    PARAMTYPE (COMPAR)  MIN 3 MAX  3        TOK (NUMBER)  (* KOMMANDO-DESCRR. *)
                                            SUP (NAMSTAR)
                                            SUP (NAMPAR)
    PARAMTYPE (PARPAR)  MIN 2 MAX  2        SUP (IDSTAR)  (* PARAMETER-BESCHR. *)
                                            SUP (NAMPAR)
    PARAMTYPE (FOLPAR)  MIN 1 MAX 10 REP 10 SUP (IDSTAR)  (* VORGAENGER-LISTE  *)
.
34  (* KOMMANDO DEFINITION                                               *)
35  (*------TYP---SCHLUESSELWORT---PARAMETERLISTE--VORGAENGER---WAHL,END,SCHLEIFE *)
.
    COMMAND 100       *BASE*       (BASEPAR) NOPARAM
    COMMAND 200       *MAIN*       (0)       NOPARAM
    COMMAND 300       *END*        (0)       NOPARAM
    COMMAND 400       *FIN*        (0)       NOPARAM

    COMMAND  10       *TOKEN*      (TOKNAM)
             PARAMETER *TYP*       (NUM)     FOLLOWING *TOKEN*
             PARAMETER *SUB*       (RANGE)   FOLLOWING *TYP*
             PARAMETER *IGNORE*    (0)       FOLLOWING *SUB*     OPTIONAL LAST
             PARAMETER *REDUNDANT* (0)       FOLLOWING *SUB*     OPTIONAL LAST

    COMMAND  20       *SUPER*      (TOKNAM)
             PARAMETER *TOK*       (SUPTOK)  FOLLOWING *SUPER*   LOOP     LAST

    COMMAND  30       *PARAMTYPE*  (TOKNAM)
             PARAMETER *MIN*       (NUM)     FOLLOWING *PARAMTYPE*
             PARAMETER *MAX*       (NUM)     FOLLOWING *MIN*
             PARAMETER *REP*       (NUM)     FOLLOWING *MAX*     OPTIONAL LOOP
             PARAMETER *TOK*       (TOKNAM)  FOLLOWING *REP*     LAST
             PARAMETER *SUP*       (TOKNAM)  FOLLOWING *REP*     LAST

    COMMAND  40       *COMMAND*    (COMPAR)
             PARAMETER *NOPARAM*   (0)       FOLLOWING *COMMAND*   LAST
             PARAMETER *PARAMETER* (PARPAR)  FOLLOWING *COMMAND*   LOOP
             PARAMETER *FOLLOWING* (FOLPAR)  FOLLOWING *PARAMETER*
             PARAMETER *OPTIONAL*  (0)       FOLLOWING *FOLLOWING* OPTIONAL
             PARAMETER *LOOP*      (0)       FOLLOWING *OPTIONAL*  OPTIONAL
             PARAMETER *LAST*      (0)       FOLLOWING *LOOP*      OPTIONAL LAST
.
    END
    FIN
```


1 - SELF DESCRIPTION OF THE DEFINITION LANGUAGE, 2 - PLACE FOR DEFINITION BLOCK OF THE FUNDAMENTAL LANGUAGE, 3 - DEFINITION BLOCK OF THE MAIN LANGUAGE, 4 - SYMBOL -- TYPE -- SUBTYPE -- NOT IN REGISTER, 5 - OPEN PAREN, 6 - CLOSE PAREN, 7 - KEYWORD, 8 - IDENTIFIER, 9 - NON-NEGATIVE INTEGER, 10 - HYPHEN, 11 - STAR NOT INTO REGISTER, 12 - STAR INTO REGISTER, 13 - SUPERTOKEN DEFINITION-COMPOSED TOKENS, 14 - SYMBOL -- TOKEN SYMBOLS, 15 - NAME, 16 - OPEN PARENS, 17 - TOKEN IN PARENTHESES, 18 - PARENS CLOSED, 19 - OPEN ASTERISKS, 20 - NAME IN BETWEEN ASTERISKS, 21 - CLOSE ASTERISKS, 22 - NAME OR PLUS, 23 - NAME IN THE PARENTHESES, 24 - DEFINITION OF CLOSED PARAMETERS, 25 - SYMBOL -- POSITION -- COUNT, 26 - TWO NAMES, 27 - TOKEN IN PARENTHESES, 28 - INTEGER, 29 - FROM -- TO, 30 - NAME IN PARENTHESES, 31 - COMMAND DESCRIPTION, 32 - PARAMETER DESCRIPTION, 33 - PREDECESSOR LIST, 34 - COMMAND DEFINITION, 35 - TYPE -- KEYWORD -- PARAMETER LIST -- PREDECESSOR -- CHOICE, END, LOOP

Fig. 6.6: Self description of the command language.

language of the definition shown. The small number can be taken as proof of the simplicity of the language. /132

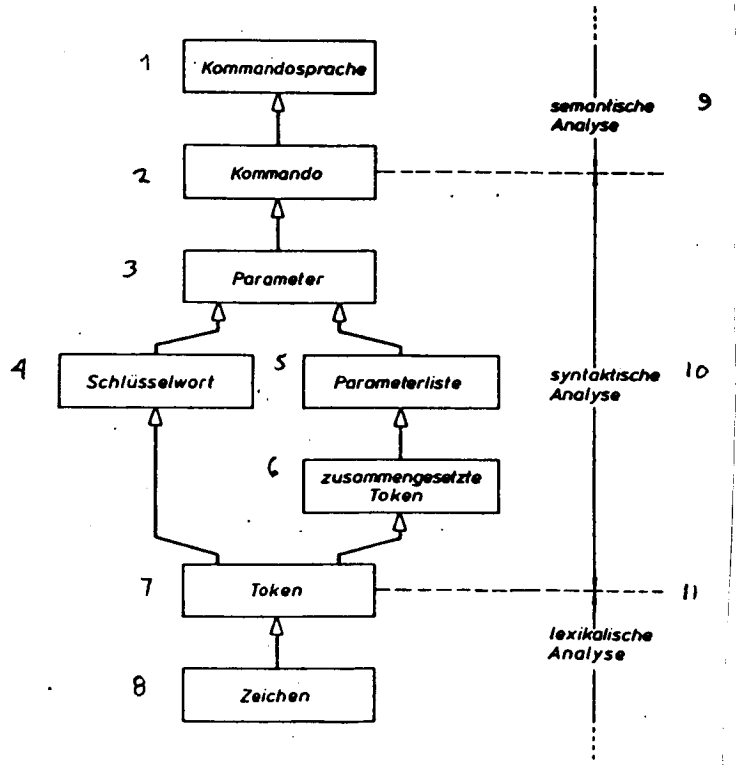
The usable tokens result from lexical analysis (compare Fig. 6.2). The types and subtypes to be declared are collected in Fig. 6.7. Here, also, a low number is characteristic. The variety results from composition of elements of this basic quantity of tokens. Thus, all closed parameter lists can be described in detail. This substantiates the many possibilities of data input as well as the exact monitoring of the input in the syntactic analysis. The examination has a decisive effect for an error free data input for the user with respect to number and order of the data. Besides the sentence forms it is above all the configuration possibilities for the parameter list on which the flexibility of the command languages rests.

Token Type	Number	Token Subtype
Text constant	1	Number of symbols
Numerical constant	2	1 : negative mixed number
		2 : negative integer
		3 : zero
		4 : positive integer
		5 : positive mixed number
Keyword	3	Value according to keyword table (created by language creator)
Identifier	4	Number of signs
Special signs	5	+, - , * etc. according to set of ASCII-symbols (compare [Mackenzie 80])

Fig. 6.7: The token types and their values (subtypes).

The order of the commands already depicts a semantic aspect of the language definition. This results from the hierarchy of the syntactic units as shown in Fig. 6.8. The region between tokens /133

and commands is the region of syntactic analyisis and thus also of language definition. The logical reference of the syntactic units described determines the order of the commands in the description. The semantic analysis of the commands is the assignment of the language creator.



1 - command language, 2 - command, 3 - parameter, 4 - keyword, 5 - parameter list, 6 - assembled tokens, 7 - token, 8 - symbol, 9 - semantic analysis, 10 - syntactic analysis, 11 - lexical analysis

Fig. 6.8: Hierarchy of the syntactic units of command languages.

The possibility to define an arbitrary number of /134 different languages is especially an advantage for the satellite system. Each segment gets its own segment language. As already mentioned, there exists for all segments the allowance to have commands for communication control (e.g. for input and output) which are the same in the entire system. These general commands are collected in a so-called basic language, whose definition must be a component of each description of a segment language. This part is set

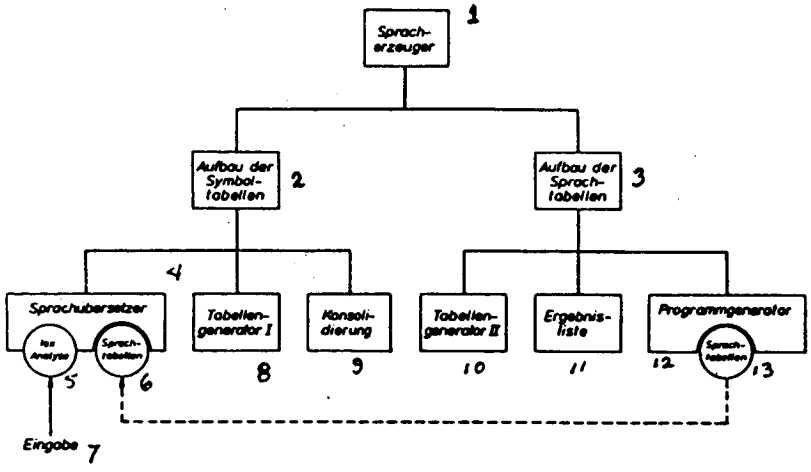
off in its own definition block (compare Fig. 6.6). This arrangement has the advantage that the description of the basic language must not constantly be newly created, rather it can be e.g. placed in each new description of a segment language as a whole block from a language library. In the language tables created out of both blocks, then, basic language and segment language appear unified.

The definition language represents a tool with which the developer of a segment of the satellite system can fix the commands for the future user. The demand that a new language be describable with the definition language guarantees the uniformity of the languages of all segments.

6.2.4 Language Creation

After the description of a language it is the task of the language creator to transform the language into the language tables. The transformation occurs in several steps as shown in Fig. 6.9:

1. Construction of symbol tables: In the syntactic analysis, the language definition is followed by the construction of the symbol tables for the intermediate symbols used (as names for tokens, assembled tokens, parameter lists, and terminal symbols (keywords). Here, for every type of /135 symbol, a table is created and processed. Following this is the semantic analysis of the definition as consolidation of the symbol tables. Here, for one thing, their completeness is checked; that is, all symbols used need to be defined. For another thing, the compatibility requirements different for each sentence form are checked (i.e. the keyword of a loop parameter may not be the same as a command keyword, since the unambiguity ofthe predecessor - successor relation would be lost). The compatibility ensures the unambiguity of a later user input with the help of the defined commands.



1 - language creator, 2 - construction of the symbol tables, 3 - construction of the language tables, 4 - language translator, 5 - lexical analysis, 6 - language tables, 7 - input, 8 - table generator I, 9 - consolidation, 10 - table generator II, 11 - result list, 12 - program generator, 13 - language tables

Fig. 6.9: Diagram with overview of the building blocks of the language creator.

2. Construction of the language tables: with this, the symbol tables are condensed, in which all intermediate symbols may be replaced by references (so-called pointers). Out of this compact form, a program generator creates a FORTRAN subprogram that contains the language table. This program can become known to the language recognizer with /136 application of the FORTRAN-EXTERNAL arrangement. After that, commands in the new language can be syntactically analyzed. In the case of self-description of the definition language there results a reproduction of the language tables (compare the dashed connecting line in Fig. 6.9). With this, it was always possible in the development of the language creator to create new language constructions with the help of old available constructions so that only at the very beginning must a simple and preliminary variant of the definition language be entered

by hand into the language table (bootstrapping [Wirth 71]). The correct working of the language creator could always be checked during the further development with the exact reproduction of the definition language.

The methods of language processing applied in the language creator owe their origin in many respects to the techniques applied in the compiliation of programming languages. As with them, a source program (i.e. language description) is transformed into an object program (i.e. FORTRAN subprogram). Here the phases of analysis (i.e. construction of language tables) will differ [Gries 71]. In the same way, the division in a lexical analysis of syntactic and semantic analysis corresponds to the procedure in compilation. Further points are the organization and processing of symbol tables, the transfer of the source program into an internal representation and the optimization of the codes to be produced (that is, condensation of the symbol tables). The last point has further the important consequence that the illustration of the language definition on the language tables cannot be reversed in such a way that the intermediate symbols used by the user could be inserted again. These are lost in the condensation. But, in any case, a depiction equivalent to the original definition can be produced from the language tables with the help of artificially produced intermediate symbols. Such an inverse process of the language creation process guarantees for the most part the operational security of the total processing of the command languages (compare Chapter 7.1).

6.4.5 Language translation /137

Using the language tables, every command can be recognized by the language which represents it. The language-independent structure of the tables means that a single language recognizer is sufficient in order to detect every command language of the class under consideration with the correct tables. It performs the syntactic analysis of any input command. The analysis is a part of the language translator, which translates any command recognized as correct into a so-called register. In this compact form, the data of further processing contained in the command are made accessible.

The language translator determines the occupation of the register for each describing command. The result of this register space assignment is communicated to the language describer and also incorporated into the language tables. Using the result list, the program developer can detect the expected position of certain data from the commands in the register input by the user. In this way the programs can be written in such a way as if the data were input in a fixed format (in the Fortran sense). This means the programs are purely oriented according to semantics. Each register consists of three parts:

1. command type: a numerical identifier for the command, which is then contained in the register. It is specified by the language describer for each command in a unique fashion (see Fig. 6.6). It is used to select the corresponding semantic program (using "computed GO TO")

2. parameter register: its content describes the present command as a sequence of its parameters. For each parameter it is specified whether it is present or not, which is important for selection parameters. In addition, which alternative for branches was selected is also specified. This information is used to perform sequence control inside the semantic program.

3. Data register. It contains all of the data of all closed parameter lists of the present command. The register locations are fixed for all lists. Figure 6.10 shows the way in which all of the data are found in the data register depending on their associated token type. /138

All of the data of the present command are contained in such a register, except for those from an open list. The reason for this is the importance of open lists for model description. The number of data varies widely depending on the model for many model properties. For such data collections, the open list at the end of the command is provided. In order to process these data, a few important additional pieces of information are required, which are located in the front command part, the so-called command head. The register represents an image of the command head. Typically it is short compared with the length of the open list.

The register has exactly enough room for one command. Immediately after the complete input of the command , it is syntactically analyzed, translated, and the register content is processed semantically. The method of operation is interpreting [Gries 71]. The interpretation of the command language has the user friendly property that both syntactic as well as semantic errors are recognized directly where they are created, and can be communicated to the user. The command re-establishment process can be used for precise error messages during /139 semantic analysis, where the register is present as an internal representation of the command. The mapping of a command to the register is reversible. Using the language tables, from the register content one can reconstruct the corresponding command in the previously input form.

1 Registerinhalt	2 Darstellung	3 Länge in Worten
4 Textkonstante	20A4 + Anzahl der Zeichen 5	21
6 numerische Konstante	-1, Real, Integer 7	3
8 Schlüsselwort	3A4 - 9	3
10 Identifikator	3A4 - 11 13	3
12 Sonderzeichen	-2, 0, Wert nach ASCII-Zeichensatz [Mackenzie 80]	3
14 Variable	-3, 0, laufende Nummer der Variablen im Makro 15 (vgl. Kap. 6.6)	3

Figure 6.10: Representation of token types and the variables in the data register (word, A4-format, real, integer are used in the Fortran sense)

1 - register content; 2 - representation; 3 - length in words; 4- text constant; 5 - 20A4 + no. of symbols; 6- numerical constant; 7 - -1, real, integer; 8 - key word; 9 - 3A4; 10 - identifier; 11 - 3A4; 12 - special symbol; 13 - -2, 0, value according to ASCII-symbol set [Mackenzie 80]; 14 - variable; 15 - -3,0, running number of variables in macro (see Chapter 6.6)

6.4.6. Functions for Language Translation

The command interpretation is taken over by the segments in the satellite system. Each section controls the command translation, in order to be supplied with new register contents command by command. However, since the translation is dependent on sections, one can design a function unit for it which can be used by each segment. The following functions are present:

1. Opening and closing of the function unit

SPRINIT

SRTERM

These functions are called at the beginning and end of a program execution. Other functions can only be used in between.

2. Loading of a new language

SRLOAD (extern)

The "external" is considered to be the Fortran type "external", and is the name of the subprogram, which is built up with the language generator and which contains the language tables of the desired language. It is only after calling of this function that sets of this language are detected and translated. This function is the only function unit which changes its internal state. All other state changes are produced by the user. /140

3. Translation of the next command

SRNEXT (ityp, lenpar, ipar[lenpar], lenreg, ireg[lenreg])

This function translates the next syntactically correct command. The register is represented by three parts: command type "ityp", parameter register "ipar", and data register "ireg". If during translation a syntactic error is detected, then an error message is output, and the language generator looks for the beginning of the next command (command key word). Semantically erroneous commands are considered to not have been input, and

they never appear in the register. An input error usually leads to a shutdown of the command interpretation. In a translation of a command the internal state of the function unit is changed by the inputs of the user. The state which can be changed in this way includes the following:

(a) the active line index in the language matrix

(b) the state of the user input (next line yes/no)

(c) the state of the present command (command type and command repetition yes/no)

(d) the state of open lists (active/inactive)

4. Interrogation of the command state

SRLOOP (loop)

For the command interpretation it may be necessary to know whether the last register content delivered by SRNEXT belongs to a command repetition ("loop" = 1) or not ("loop" = 0) (see Chapter 6.3.).

Figure 6.11 shows the calling sequence diagram of the function unit. The great dependence of translation on the semantic analysis is emphasized by the low number of functions. /141

Since the general commands of the basic language are integrated in each segment language, the corresponding semantic programs also have to be made available.. However, in order to not have to include this in each section and to overload the program text several times, they are all summarized in a layer between the section and the language translator. This means an expansion of the function unit discussed above by the function

COMAND (ityp, lenpar, ipar[lenpar], lenreg, ireg[lenreg]).

Instead of SRNEXT it is to be used for command interpretation in the next sections. This function itself uses SRNEXT and if a general command is being translated (detectable from

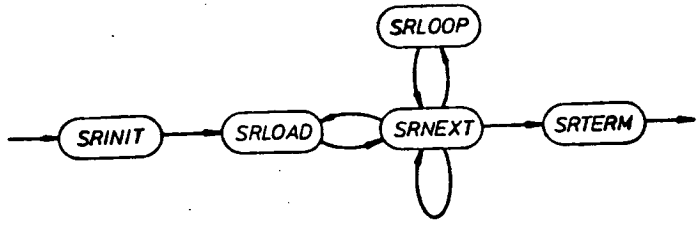


Figure 6.11. Sequence diagram of the functions of the language translator

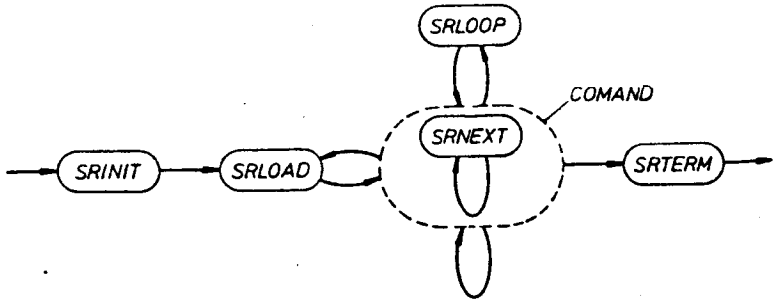


Figure 6.12. Calling diagram of the functions of the language translator with consideration of transparent commands

the command type), then the corresponding semantics program is called directly. In this way COMMAND only gives the translation of section-specific commands to the next section. Since the COMAND acts as a sifting function, so that for the sections the general commands are not visible, these are also called transparent commands. In this way we obtain the modified sequence calling diagram of Figure 6.12.

6.5. Open Lists /142

6.5.1. Concept

Open lists have a special position within the command language in several ways:

1. Numbers are exclusively the object of description in open lists.
2. Most model data are represented by numbers and are primarily described in open lists.
3. The syntactically unlimited length of open lists allows the description of arbitrarily large data amounts.
4. The syntax of the open lists differs from that of the command head. It is not producible by the language generator, but it is the same for all open lists of each command language (see Figure 6.13).

An open list is defined as a line by line enumeration of a table with a fixed number of columns, and an open number of lines. If the number of columns is n, then a line of the table is called an n-tuple, and the list is an open sequence of n-tuples. The tuple length n is an invariant of the open list. The concept of the table was introduced in Chapter 5.3 as a data module for storing model data. In this way the open lists are the language items with which the user fills the tables with data.

In this connection, we should mention that the correspondence of the table and the open lists corresponds to the correspondence of the control list and the command head. This contains the (type)-names of those data in the control list which are required for processing the elements of the open list. This is a profound reason for the pragmatic division of commands into a command head and the open list.

According to the tuple length n, two types of open lists are distinguished:

- n = 1 :simple (open) list
- n 1 :tuple list

Syntactically, both types are formed in such a way that a simple list represents a sequence of numbers. In the case of tuple lists, the tuple are to be separated by means of a separator, a special symbol (symbol:/). The table character of tuple lists can be made visible by the user by a corresponding arrangement of numbers on the input lines. The simple lists are of two different forms:

1. ORDINAL: This key word briefly describes the list

1, 2, ... m

where m is a semantic aspect of the list, which is derived from the corresponding command and the prevailing network tree.

2. ALL: This is an abbreviation for all values of a /144 numerical collection, whereas the elements of the collection are specified in a different way. For example, they are specified by a simple list of another command.

6.5.2. Functions of list generation

Since during command interpretation, the same operations are to be performed for each element of an open list and the number of elements depends on the model, one requires a tool for reprocessing of open lists element by element. One of

```
<open list>        ::= <simple list> / <tuple list>
<simple list>      ::= <numbering> / ordinal/all
<tuple list>       ::= <tuple list>/<tuple>/<tuple>
<numbering>        ::= <numbering><numerical constant>/ <numerical constant>
<tuple>            ::= <tuple><numerical constant>/ <numerical constant>
<numerical costant>:   see Figure 6.2
```

Figure 6.13. Numerical constant

the tasks of a functional unit conceived for this is the syntactic analysis of lists and the determination of the next number. The lexicon analysis is used which is the same as for the language translator. The functional unit is used by the semantic programs, in order to produce a pure number sequence. They are also called list generators. Since the processing of the command head and of the open list is done one after another, the language translator and the list generator mutually exclude one another: while the language translator is active, the list generator cannot be called ,and vice versa. This is considered in the calling sequence diagram of Figure 6.14. The following functions are provided for list generation:

1. Beginning of an open list

LSETUP (lentup, iordal)

These functions displace the list generator into the active state. The expected tuple-length "lentup" is a semantically determined quantity, and therefore has to be specified again for each open list. If it is a simple list ("lentup" = 1), then depending on their special form we have:

```
ORDINAL : 'iordal' = 1,
ALL     : 'iordal' = 2.
```

In all other cases, we always have "iordal" = 0. Whether /145 a special form is possible has to be examined by the semantic analysis.

2. Next numerical value in the list

LSNEXT (iend, next, fnext)

This function is called exactly once for each element of an open list. For a simple list in one of the mentioned special forms, the function is not called. The result is:

"iend" = 1 The produced numerical value is the last one in the list. The beginning of the next command follows, or a loop parameter of the present command.

"iend" = -1: An input error in the list was discovered. A corresponding error message is given to the user directly by the list generator. A further calling of the function is not allowed and would also not make sense.

"iend" = 0 : There is no input error, and the list end has not been reached.

"fnext" : Representation of the value as a number of the Fortran type "real".

"next" : Representation of the value as a number of the "integer" Fortran type:

```
next = div (fnext+o.5,1.)
```

If the numerical value cannot be represented in this way because its size goes beyond the representation possibilities of the physical memory locations, then "next" is given the value zero.

3. End of an open list

LSUPST

This function concludes the functional unit. This is related to a reactivation of the language translator, which after this can be called again. If during processing of the open list an error occurred, either of the syntactic ("iend = -1) or the ematic type, then no end of list has yet been reached. The function therefore looks for the end of the list and in this way allows a subsequent calling of SRNEXT (Chapter 6.4.6) without errors.

The language translator and the list generator are the tools with which one can process the language side of the model description. Figure 6.15 shows how this is done in a unified way which can be used for all command languages. The program sequence plan includes all functions of the two functional units.

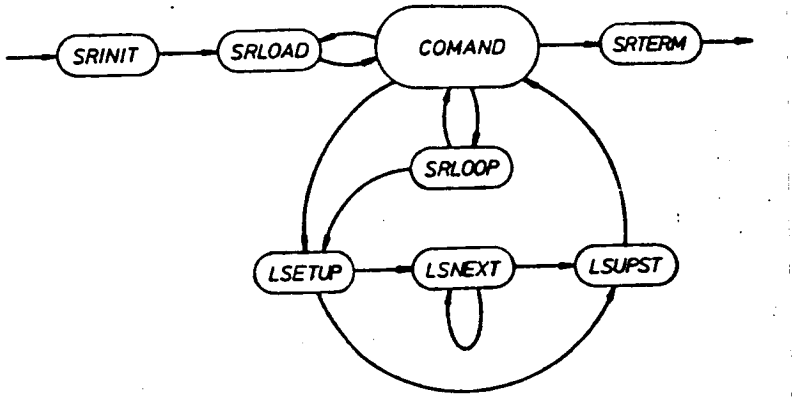


Figure 6.14. Calling sequence diagram of functions of the language translator and the list producer

This diagram can be considered as a frame for all semantic programs of the satellite system. /148

6.5.3. List description

Usually the user will be anxious to arrange his model data in a regular form in order to facilitate data input. Therefore in open lists finite numerical sequences will often occur, which can be specified in terms of a formation law. For each pair of successive terms of a numerical sequence with n terms we have:

$$a_{i+1} = a_i + \Delta a_i \qquad (6.1)$$

Here and in the following we will have $1 \leq i \leq n.$. A formation law is now established, if a rule $f(\Delta a_{i-1})$ is used for determining Δa_i , so that we have

$$\Delta a_{i+1} = f(\Delta a_i) \qquad (6.2)$$

For the simplest and most frequent of the numerical sequences of this type, we shall now introduce a language construction, which will allow the user to describe very long numerical sequences

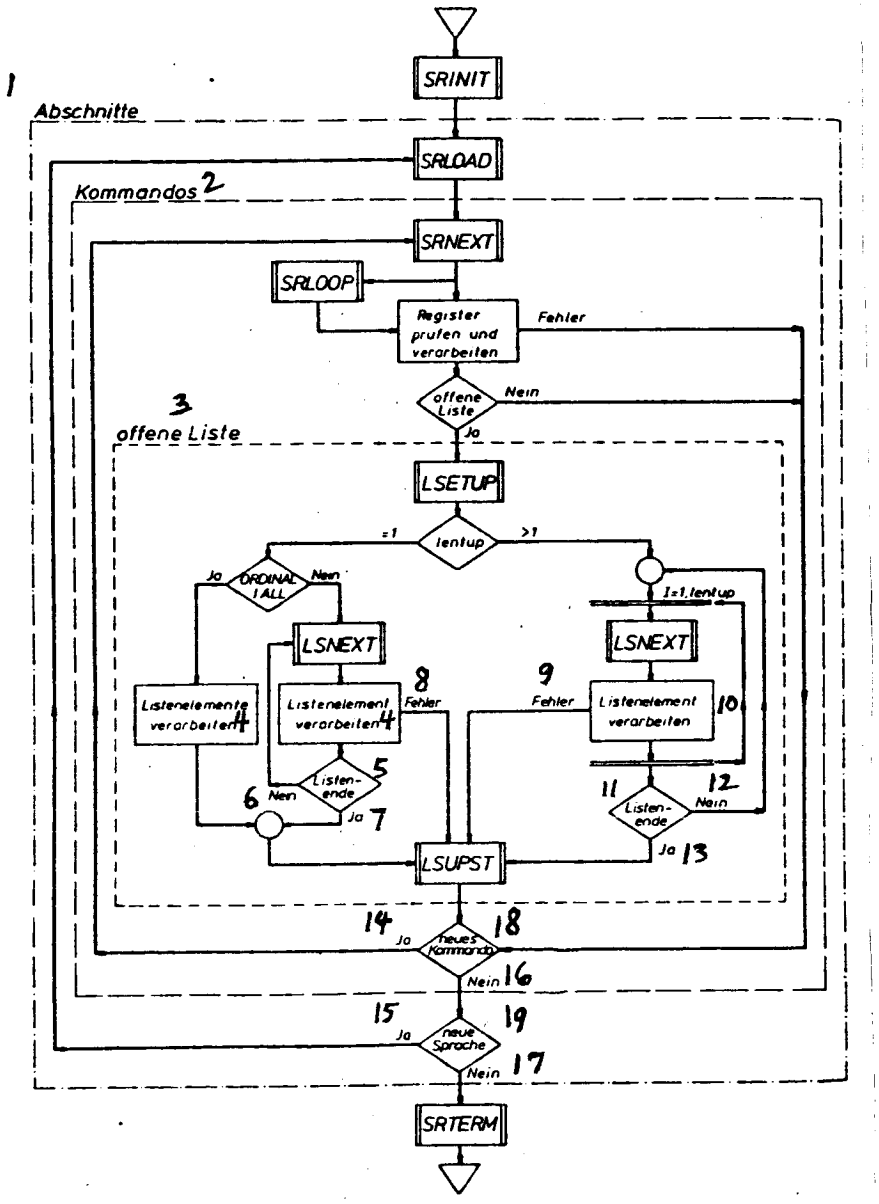


Figure 6.15. Schematic program sequence plan of language processing (Please see key on following page)

Key for Figure 6.15. 1 - section; 2 - command; 3 - open list; 4 - list elements; 5 - end of list; 6- no; 7 - yes; 8 - error; 9 - error; 10 - process list element; 11 - end of list; 12 - no ; 13 - yes; 14 - yes; 15 - yes; 16 - no; 17 - no; 18 - new command; 19 - new language

with only a small amount of data. The complete syntax is given in the syntax diagram of Figure 6.16. The following numerical sequences can be distinguished:

I. First degree numerical sequence

1. Constant sequence: This is a numerical sequence which remains the same

$$\Delta a_i = 0 \,. \tag{6.3}$$

Its language form is:

FROM <a_1> LOOP <n> CONST REP

2. Arithmetic sequence of the first order: Its formation law is

$$\Delta a_i = \Delta a_1 = \text{const} \neq 0. \tag{6.4}$$

From the determination variables a_1 and Δa_1 , we can directly determine all the members of the sequence:

$$a_i = a_1 + (i-1) \cdot \Delta a_1 \tag{6.5}$$

/149

Its language form is:

FROM <a_1> LOOP <n> { BY <Δa_1> / NEXT <a_2> / TO <a_n> } REP

If Δa_1 is not given explicitly, then it can be determined from

$$\Delta a_1 = a_2 - a_1 \quad \text{(for NEXT)}$$

$$\Delta a_1 = \frac{1}{n-1}(a_n - a_1) \quad \text{(for TO)}$$

3. Arithmetic sequence of the second order: The represent an arithmetic sequence of the first order, so that the formation law is

$$\Delta a_i = \Delta a_1 + (i-1)d \tag{6.7}$$

with d = const $\neq$ 0. The determination variables of the sequence are a_1, Δa_1, d , and the terms are determined by

$$a_i = a_1 + (i-1)\Delta a_1 + \frac{(i-1)\ (i-2)}{2}\ d \tag{6.8}$$

This sequence can be described with

$$\text{FROM } \langle a_1 \rangle \text{ LOOP } \langle n \rangle \begin{Bmatrix} \text{BY} & \langle \Delta a_1 \rangle \\ \text{NEXT} & \langle a_2 \rangle \\ \text{TO} & \langle a_n \rangle \end{Bmatrix} \text{ DIF } \langle d \rangle \text{ REP}$$

When a_n is specified, then Δa_1 is found from Equation (6.8) as

$$\Delta a_1 = \frac{1}{n-1}\left[a_n - a_1 - \frac{(n-1)\ (n-2)}{2}\ d\right] \tag{6.9}$$

4. Geometric sequence of the first order. This is a sequence with the formation law

$$\Delta a_i = \Delta a_1 \cdot r^{(i-1)} \tag{6.10}$$

where $r = (1 + \frac{\Delta a_1}{a_1}) = \text{const} \neq 1.$. The determination variables of such sequences are a_1 and Δa_1 , and we have /150

$$a_i = a_1\ (1 + \frac{\Delta a_1}{a_1})^{(i-1)} \tag{6.11}$$

5. Geometric sequence of the second order. Similarly to the arithmetic sequences, the Δa_i make up a geometric sequence of the first order

$$\Delta a_i = \Delta a_1 \cdot r^{(i-1)} \tag{6.12}$$

with $r = \text{const} \neq (1+\frac{\Delta a_1}{a_1})$ and r $\neq$ 1. The sequence has the determination variables a_1, Δa_1 , and r and the determination equation is

$$a_i = a_1 + \Delta a_1 \frac{r^{(i-1)} - 1}{r-1} \tag{6.13}$$

Both types of geometric sequences are described by

$$\text{FROM } \langle a_1 \rangle \text{ LOOP } \langle n \rangle \left\{ \begin{matrix} \text{BY } \langle \Delta a_1 \rangle \\ \text{NEXT } \langle a_2 \rangle \\ \text{TO } \langle a_n \rangle \end{matrix} \right\} \text{ RAT } \langle r \rangle \text{ REP}$$

where for $r=(1+\frac{\Delta a_1}{a_1})$ there results a geometric sequence of the first order and otherwise a geometric sequence of the second order. If a_n is specified, then according to equation (6.13) we have

$$\Delta a_1 = \frac{r-1}{r^{(n-1)}-1} (a_n - a_1) \qquad (6.14)$$

II. Second degree numerical sequences

If a numerical sequence of the first degree is run through several times and if its initial value changes according to one of the form formation laws and if in addition all of the other determination variables remain constant, then a second degree numerical sequence results. It is described just like a numerical sequence of the first degree. For the description there is a numerical sequence of the first degree (with LOOP n ...) and there is a description of its repetition and the formation laws to be followed (with AND m ... see Figure 6.16). In order to be able to reduce numerical patterns which are different /151
from "rectangular patterns", one also provides for a change in the number of following terms according to an arithmetic or geometric series. This is described by means of DIFLOOP or RATLOOP (see examples of Figure 6.17).

III. Numerical sequence of the third degree

Just like the formation of the second degree numerical sequences, from them the numerical sequences of the third degree are produced by a change in the initial value according to some law. Accordingly, the description of a

second degree numerical sequence (with LOOP n ... and AND m ...) is complemented by specifying the number of repetitions and the formation laws to be used (see AND 1 ... in Figure 6.16). In order to be able to produce numerical patterns different from a prismatic pattern, the change of the following terms according to arithmetic or geometric series is provided for. This change is described for the numerical sequences of the first degree by DIFLOOP or RATLOOP and for the numerical sequences of the second degree it is described by DIFAND or RATAND, respectively.

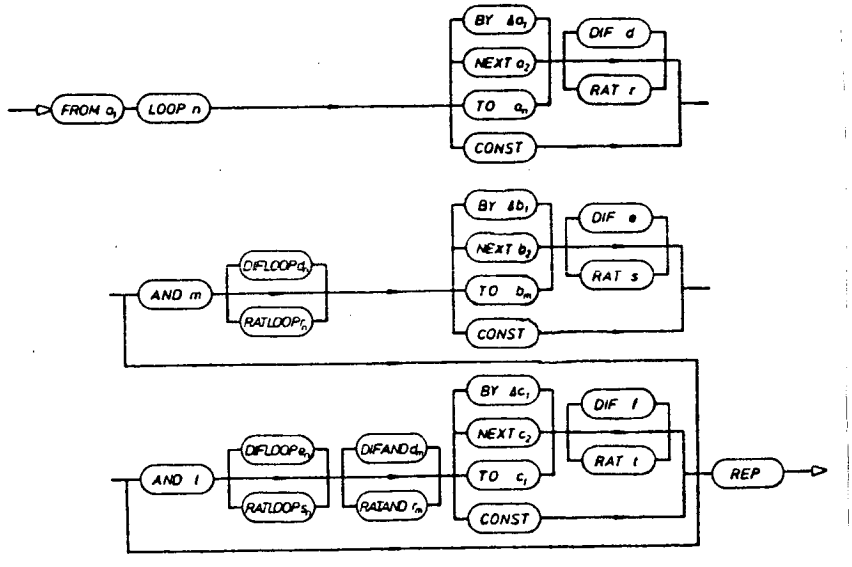


Figure 6.16. Syntax diagram for describing numerical sequences

All of the mentioned numerical sequences can be expanded by tuple sequences. The mentioned formation laws are used again on all of the tuple elements in the same way. /152

The syntactic analysis of the sequence description is the task of the functional unit of the List generator. This determines also a sequence of the next value in the list (function LSNEXT) from the specified determination variables. For the series sequence of the values we have the following priority diagram

1. Tuple
2. First degree sequence
3. Second degree sequence
4. Third degree sequence

In order to explain the description possibilities, Figure 6.17 gives a few examples of numerical sequences.

```
 1,  2,  3,  4,  5,  6,  7,  8      ≙   FROM 1 LOOP 8 BY 1 REP

 1,  2,  4,  7, 11, 16, 22, 29      ≙   FROM 1 LOOP 8 BY 1 DIF 1 REP

 1,  2,  4,  8, 16, 32, 64,128      ≙   FROM 1 LOOP 8 BY 1 RAT 2 REP

 1,  2,  3,  4,  5,  6,  7,  8,
 9, 10, 11, 12, 13, 14, 15, 16,     ≙   FROM 1 LOOP 8 BY 1
17, 18, 19, 20, 21, 22, 23, 24,            AND  4 BY 8 REP
25, 26, 27, 28, 29, 30, 31, 32

 1,  2,  3,  4,  5,  6,  7,  8,
 9, 10, 11, 12, 13, 14,             ≙   FROM 1 LOOP 8 BY 1
17, 18, 19, 20,                            AND  4 DIFLOOP -2 BY 8 REP
25, 26

 1,  2,  3,  4,  5,  6,  7,  8,
 9, 10, 11, 12, 13, 14,             ≙   FROM 1 LOOP 8 BY 1
15, 16, 17, 18,                            AND  4 DIFLOOP -2 BY 8 DIF -2 REP
19, 20

 1,  2,  3,  4,  5,  6,  7,  8,
 9, 10, 11, 12,                     ≙   FROM 1 LOOP 8 BY 1
17, 18,                                    AND  4 RATLOOP .5 BY 8 REP
25

 1,  2,  3,  4,  5,  6,  7,  8,
 9, 10, 11, 12,                     ≙   FROM 1 LOOP 8 BY 1
13, 14,                                    AND  4 RATLOOP .5 BY 8 RAT .5 REP
15
```

Figure 6.17. Examples of number sequences of the first and second degree

6.6 Macrocommands /153

6.6.1 Properties

The previously introduced command language has the following characteristic properties:

- For each command the interpretation follows directly after translation: interpretative command processing

- The commands are processed in the same sequence as with which they were input: sequential command series

- The elements of the parameter lists explicity contain values: specified command content

These properties are independent of whether or not the input was in the form of individual commands (on the screen) or as a command stack (from an operating system data bank). They are no longer sufficient for model description, for example, when it is necessary to input a series of command with slightly changed data several times, which can occur for very similar models (network generator key word). The comfort of the command language for such applications can be substantially increased by introducing so called macro commands or macros for short. By macro we mean a sequence of translated commands which are identified by names selected by the user (see [Cole 76]). The data registers and open lists contain three locations, which are filled up by arguments specified when a macro is called. The substitution of the arguments is done by command. After this, the complete command is available for interpretation. The process of substitution of arguments and the making available of commands is called (macro) expansion. The macros have

the advantage of being able to be used arbitrarily with any other arguments. Also, the interpretation process is accelerated because the translation step is not needed.

It is remarkable that the capacity of the command language to form macros follows very naturally from the strict separation of syntax and semantics or of translation of the commands. The flexibility in the use of the satellite system is clearly visible in a number of phase transitions during command processing as shown in Figure 6.18. /154

The introduction of macros means an expansion of the language elements in two ways. First of all, the commands defined in the language description are supplemented by the macro commands defined by the user himself. In addition, so-called pseudo commands apply within the macros. These are instructions to the macro expander, which allows the joining of variables, operations on variables and the control of the command sequence during expansion.

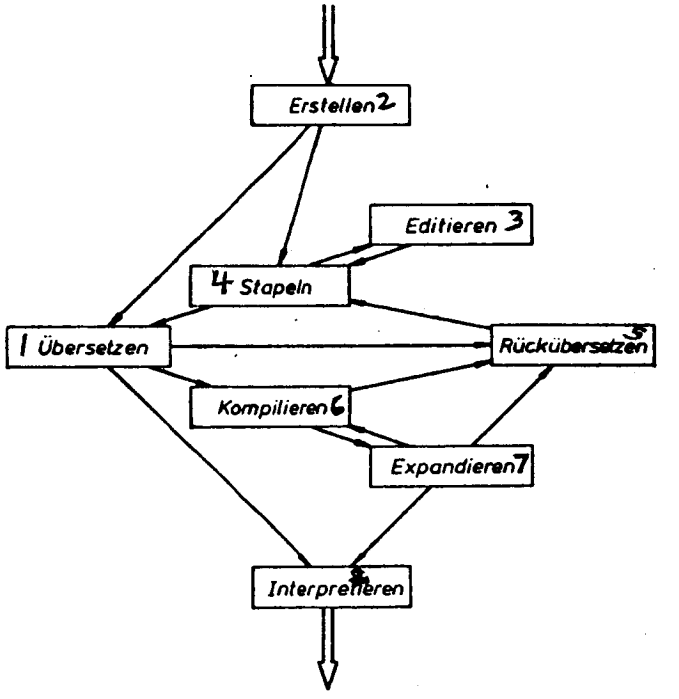


Figure 6.18. Phases of command processing 1--translate; 2--set up; 3--edit; 4--stack; 5--reverse translate; 6--compile; 7--expand; 8--interpret

This amounts to introducing algorithms in the command language. /155
The pseudo commands are effective during expansion, and depending on the intents of the user, a stream of (model description) commands are produced, which no longer contain pseudo commands. The command language expanded in this way is then characterized by the following properties:

- Translation and interpretation of commands are done at different times: compiling command processing

- The sequence of interpretation of commands can be influenced based on selected conditions: conditional command sequence

- For the translation, not all the values of the parameter lists have to be specified: variable command content

6.6.2. Macro description

Based on the separation of translation and expansion of the macros, two phases are also introduced when they are applied. First of all, a macro has to be described. This is done in three parts:

1. Macro convention: The macro is given a name it is determined on which data bank the macro is to be compiled. Then the macro description is deposited there in the translated form.

2. Variable convention: If the subsequent commands contain variable command contents, then names have to be assigned. The variables in addition are given a class and a type.

3. Macro body: This is all of the commands, which are to be expanded during a call of the macro. This includes commands for model description from the active command language, macro calls and pseudo commands.

/156

Figure 6.19 gives the syntax diagram of macro description. An important concept of the macro is the treatment of the variables. The concepts of macro environment and command step play a special role here. The calling of a macro is possible by the user during command input (command level 0), or this can take place from another macro (command step >0). The command step (i) to be called represents the surroundings of the called macro (command step i+1). One of the three following variable classes is assigned to each variable (see [Dijkstra 76]).

1. Surrounding-independent variables (local variables): The name of such variables is only known inside the corresponding macro and its value is only accessible there. The production and change of the value is done using pseudo commands. Before the first generation, the value is undefined. Each operation with undefined values is disallowed accept for an assignment of a defined value.

2. Call-dependent variables (arguments):

   Its name is also known in the surroundings. The initial value is set when the macro is called. It is, therefore, known outside of the macro. After this call a variable like a local variable is processed. A change in the value during expansion is not felt in the macro surroundings. In other words, the initial value remains intact there.

3. Call dependent variables (global variables): Its name and value are known in the macro surroundings. A change of these variables produces a change in the surroundings. For this reason, such variables are not allowed in the macros of command step zero. In every case, only the used variables from the surroundings have to be considered as global variables according to convention. Global variables always lower command step than local variables.

In addition to the assignment to a class, all variables are also of a certain type. /157

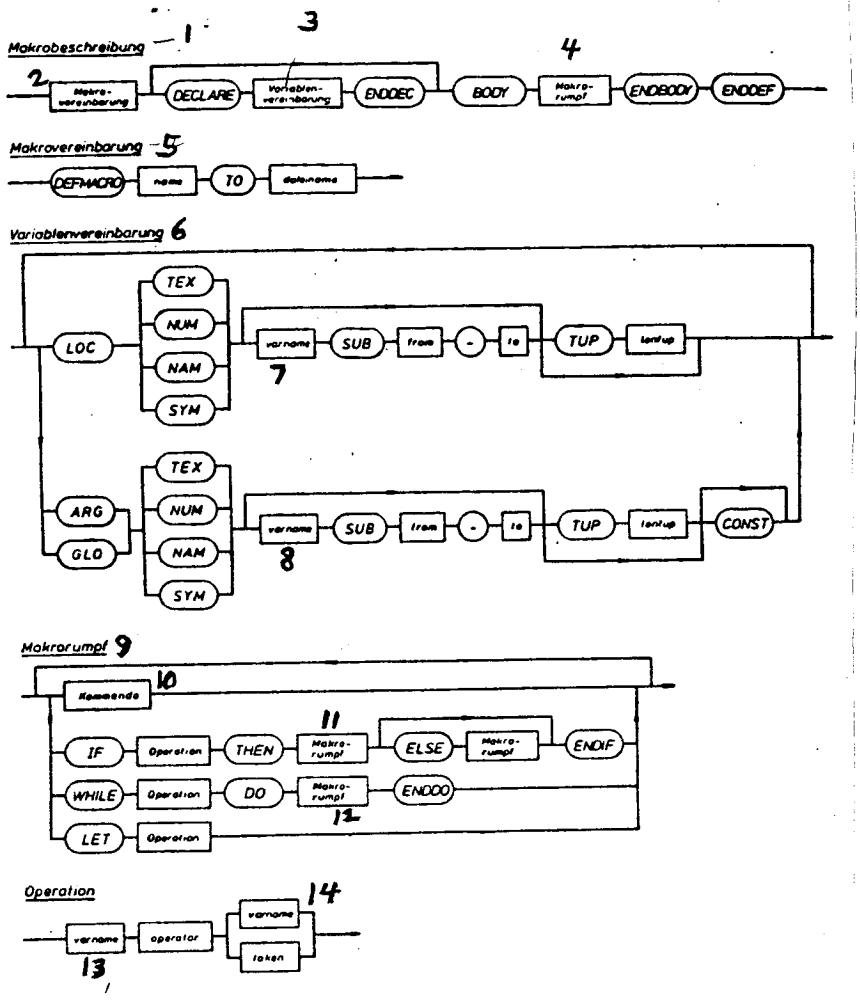


Figure 6.19. Syntax diagram of the macro description
1 - macro description; 2 - macro convention; 3 - variable convention; 4 - macro body; 5 - macro convention; 6 - variable convention; 7 - first name; 8 - first name; 9 - macro body; 10 - command; 11 - macro body; 12 - macro body; 13 - first name; 14 - first name

This specifies the memory form and the possible operations. The variable type is specified by four attributes: /158

1. Token type: This is the type of the token, which represents each variable.

2. Value range: This is the range of possible token subtypes (see Figure 6.7), within which the values of the variable can lie. The largest possible value of the corresponding token type is the range adjusted beforehand.

3. Tuple length: Simple variables can stand for a value in all parameter lists (tuple length 1). Tuple variables can only be substituted for a single entire tuple within open parameter lists. (Tuple length >1). Simple variables are set first.

4. Constant nature: If the value of a variable should not be changed by pseudo commands, then it is constant. This means that during the expansion the use of variables is monitored, so that it is impossible to overwrite the value. Since local variables do not have a defined value at the beginning of expansion, it cannot be constant. The constant nature of the variable is not set before hand.

The storage forms of variable values correspond to the representation of the associated tokens in the data register (see Figure 6.10). As general operations, the following are possible for all variable types

- the association (operator EQU) and
- the test for equality (operators EQ, NE).

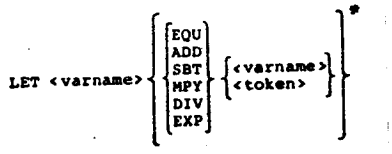
In addition, there are special arithmetic operations which are only allowed for variables of the token type number:

- calculation operations: ADD (+), SBT (-), MPY (*), DIV (/), EXP (↑).
- comparison operations: LT (<), LE (≤), GE (≥), GT (>).

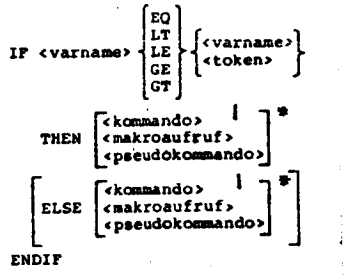
All of these operations are only triggered by pseudo commands:

/159

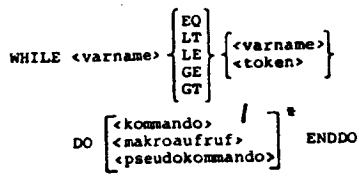
- association and calculation operations:



- test for equality and comparison operations:
  Branching



  Pre-tested group



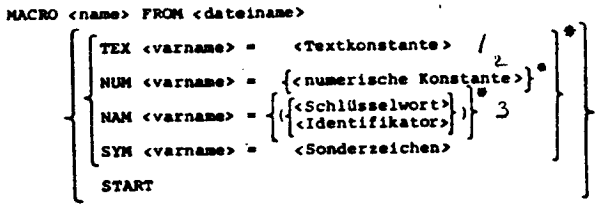
1--command, macro call, pseudo command

The pseudo commands are commands in the sense of a command language. They represent instructions for the expansion process. Semantically, several pseudo commands can have a composite structure, such as branching and pre-tested groups. They are known as standard structures from programming languages [Linger 79]. In this way, one can realize the conditional command series sequence without using jump instructions.

/160

The variables agreed upon by convention can be substituted for each element of a parameter list within a command sequence of the macro body, if there is compatibility with respect to type. For this, instead of the value, one specifies the variable name, which is specified within two dollar signs $varname$. The special symbols are for the language translator for detecting the variable. The special representation of the variables in the data register is related to this (see Figure 6.10). In this way, one can have a convention regarding variable command content in the macros.

All of the commands of the macro body are translated and the registers are stored together with the variable conventions on the fixed data bank. If open lists occur, then their syntactic units are also stored in the data bank. The macro data bank contains all the information of the commands contained in the macro body with exception of the argument values. These are specified in a macro call which has the following structure:



1--text constant; 2--numerical constant; 3--key word, identifer, special symbol

The syntax for the intermediate symbols of the right side of the argument associations can be taken from Figure 6.2. For all of the agreed upon arguments, and only for them, the values selected have to be specified for each macro call. In this way during the expansion of the macro, one obtains a sequence of commands which can be directly interpreted. Appendix A gives an example of a description and a call of macros.

7. Aspects of a user friendly model description /161

7.1 Principles of communication

The design of a command language was done with the intent of giving the user a convenient tool for model description. The principles used will be summarized in the following.

Language is especially useful for communication between humans. In a very simple form of the command language, among engineers it can be used for communicating a certain model. The wide use of names and the abbreviated description of model data in open lists result in a readable and clear picture of model description. In this form, the transfer of a model description through space and time is easily possible without any loss of any understandability. As far as the understandability among engineers is concerned, this should also apply for communication with the satellite system. In the language translated, an automatic method for transferring commands into a program-like form should be found, which does not have to be specified by the user himself. This leads to the principle of language communication:

- Every communication of a user with a satellite system occurs exclusively with commands of the language which is specified by the given syntax (Chapter 6.3).

This does not exclude all of the side paths, which may be used because of expediency. (For example, for generating a simplified input in user programs). However, this does not then guarantee the complete readibility of every model description.

Because of the strict separation of syntax and semantics, in addition to the command language, it would also be possible to have a dialog oriented communication in the form of question and answer method, or using a menu technique. Both methods, however, are not suited for communication among engineers or for documentation of a model description. In addition, stap processing becomes impossible. Therefore, /162
these communication methods are now considered first in the model description.

In order to ensure that the communication always occurs completely and uniquely, that is, there is no loss of important information, the principle of reversibility of language translation has to apply. This means that a function unit exists, which can produce a command from the content of each register using the language tables, which is equivalent in its action to the one input before by the user. Only the information which does not have syntactic importance is lost, which is related to the free format of input (see Chapter 6.2). The commands are mainly used for describing data. This means we have the following

- language description is used for producing tables
- list description is used for producing extensive numerical series
- macro description is used for producing a command stream
- the model description is used for producing the model data in the project data bank

Therefore, there is an analaqous correspondence of description and the producing system. By expanding the last mentioned principle, therefore, we have to specify the principle of reversibility of communication. This means that during communication from user input until a data point is produced in the inner state of the satellite system, no important information is lost. This means that always one can automatically produce a stream of description commands from the data using the reversal method, which is equivalent in its effect to the previously occurring user input function. This means that there is a describing system which can be generated for each producing system, which provides for self-documentation of any describing data structure.

/163

7.2 Data generators

For the user friendly model description, data generators are absolutely necessary. In most models, regular distributions can be found for some model data, which are suitable for being described with relatively small amounts of data. In individual cases, much data input can be saved. The command language makes available two general tools for this:

- The list description for abbreviating arbitrary regular numerical sequences (Chapter 6.5).

- The macro description for the easy and variable repeatability of entire model descriptions or parts (Chapter 6.6).

Any other aid for data generation has to follow the principle of name specification and name generation (Chapters 5.2.1 and 5.3.1). In contrast to the procedure for so-called net generators (see [Pfaffinger 81]), we explicitly dispense

with the program production of nodes and elements. The user identifies every node and each element using a number which he selects, which has to be agreed upon before it is used the first time. The same is true for networks and load cases. Additional general aids include:

- Describing model data can be copied between various net contexts for the same network type and the same node number. This means one of the two descriptions becomes superfluous.

- Among the various elementary networks, model data with the same model type and possible with the same model or load type can be exchanged for all elements or for one of them.

- All measurement variables can be described by a free selection of scale units in a useful way (for example, coordinates, loads).

/164

Additional generators and aids for data generation depend on the class of model data (see Chapter 3).

- Bundling of elements of an elementary network for the simplified incidence description (for example, two triangle elements to make up a square).

- Selection of suitable coordinate systems for the coordinate description.

- Selection of suitable bases for the description of the configuration.

- Use of the principle of complete coordinate description (Chapter 3.6)

- Interpolation of coordinates on the basis of topological regions.

- Superposition of load cases.

The mentioned data generators are controlled by work and element properties which can be parameterized (see Chapter 5.2). In addition, generators for element data and element loads are to be provided for, which for the most part, can be characterized by element properties which cannot be parameterized. This raises the question of the work division between generators for such data and the programs which process them and to perform element calculation. The general nature of the data to be produced can be used as guidelines for conceiving elementary data generators. This occurs when many model or load types have a single common property, for example

- cross section data for different beam elements,
- reference systems for the orientation of material characteristics,
- different material laws such as for isotropy, orthotropy or certain crystal structures,
- temperature and speed distributions.

/165

On the other hand, the model type and the load type allow a selection of elementary properties and the required data (see Chapters 3.9 and 3.10). In this way, by input of a small amount of simple data, one can provide for sufficient element description. As examples we can mention

- using the element data density and the network data angular rate and rotation center, all centrifugal loads are completely described,
- with the element data of expansion coefficients (and their reference system in the anisotropic case) as well as the network data for temperature distribution, all temperature-cause initial expansions can be described.

Therefore, the element programs should be designed carefully for the user and there should be a large variation of model and load types available. In this way, the element programs can take over important functions of data generation.

7.3 Errors

According to a naive point of view, data generators are the most important part of the system for model description. There is a desire to support the user during the production of extensive amounts of data. This point of view is naive because it uses the unrealistic assumption, that the data described in this way is always correct. The converse is true: Most models first will have errors. This does not mean that large data generators which take over most of the tasks of data production do not have any value. Instead, this expanded point of view gives support of the user in the task which is usually the most difficult: Finding errors and eliminating them.

The following error sources have to be considered:

1. Errors in the model description: Description errors. /166

2. Errors in the modeling of the physical reality. The describing model does not agree with the one to be described: idealization errors.

3. Errors in the programs of the satellite systems: Programming errors.

Whereas the user is effected by all three types of errors, the satellite system can only discover description errors itself actively. How important such a discovery of errors is can be made clear by considering some of the consequences:

- Errors in the model description can lead to severe non-correctable errors in the operation of the core system: no results

- Errors in the model description can only be detected by careful analysis of calculated results: erroneous results

- Errors in the model description are not noticed at all, that is the calculated results do not apply for the model under consideration: apparently correct results

The concept for treating description errors includes four phases:

1. Prophylaxis: First it is important to avoid errors. This conceals the complex relationship between the system and the user, in which psychological factors play an important role (see [Weinberg 71]). All aspects of the method of working with the system can lead to avoiding errors. This includes the principle of language communication, the free input format and the list description. The most important factor is the extensive information of the user which can be done as a preparation or in parallel with the model description. This makes use of the following:

- Aid functions: The user can obtain a display of the possible commands (especially on the screen) and

can find out about the structure, content and meaning of the individual commands.

- Summary functions: The user can become informed about the status of the model description (see Chapter 4.1) and can obtain a list of the input model data in a table form.

- Handbooks and training courses: When the portion of instructions for the user, especially the beginner, can be made very small, then one can evaluate the system to be user friendly.

2. Analysis: Since one can assume that errors are present in the model data, these have to be found. This should be done as close as possible to the error source, the input of erroneous data. In this way, errors are better localized and their creation becomes more understandable to the user. In addition to syntactic errors, contradictions and incompletenesses in the input data have to be discovered. Only such logical errors can be found at all from the program. On the other hand, physical errors cannot be detected, which results from the fact that the described model does not correspond to the one which is being described. In summarizing, we can say:
   correct model = physically correct idealization
   + logically correct model description.

The following tests are used for error detection:

- Input tests: The content of each syntactically correct command is tested for plausibility. Only syntactically and semantically correct commands are processed. Logical errors are discovered, which result exclusively from the

command content.

- Completeness tests: Gaps in the model description are discovered.

- Compatibility tests: Within the framework of section consolidation, the input data are related to one another. Errors which cannot be found in individual data are discovered (see Chapter 4.1). The effort /168
for error analysis must not be very great, if the logical correctness of a model description has to be insured. The user only in this way will be given sufficient competence for interpreting the calculated results.

3. Diagnosis: Errors found have to be described sufficiently. The description has to be easily understandable for the user and has to be complete enough in order to allow an easy detection of the error source (see [Horning 79, Parsons 79, Shneiderman 82]). Diagnosis includes the following:

- Error stage: This determines the weight of the error and the required reaction by the user:
  - 0: Communication, intended to orient the user.
  - 1: Indication of possible errors. The user should pursue these possibilities, because there is the danger of subsequent errors.
  - 2: Errors, which have to be eliminated at all costs.
  - 3: Programming errors, for which the user is not responsible.

- Error number: This is used for identification of the error. Then using an error handbook with auxiliary functions, one can find an extensive description of the error.

- Error communication: The error is described in a natural language in a polite form. Positive expressions and the use of concepts from the world of the user can have a psychological positive effect: An error found indicates a false model description and not the bad intentions of the user.

- Additional information: Using numerical and alpha-numerical values, the error is determined more accurately. The values are basically only known at the time at which the error is detected, so that for the same error number they can always change. For example, this includes the discovered erroneous data.

/169

4. Therapy: The user has to eliminate the error found. This means that he does not only have to understand the error, but also has to find the error source and must correct the erroneous data. This requires different procedures which depend on the error. The error number is suitable for requesting information as to how the error is to be corrected (using auxiliary functions or the error handbook). This information includes:

- An explanation as to what went wrong, if it has not already been described by the error message.

- An indication how erroneous data is to be found (for example, by listing).

- A procedure for correcting the data.

Idealization errors are very difficult to detect and can only be avoided by the user by a very careful model description.

Reliability is a very important property for evaluating the user friendliness of the satellite system. If programming errors occur nevertheless, which is demonstrated by a surprising behavior of the system, then they must be capable of being eliminated quickly. The following is a list of prophylactic measures against programming errors:

- A clear and clean modelling of the program system into functional units. In this way, errors can easily be found.

- A basic testing of the transferred parameter values in subprograms, in order to insure correct use.

- A testing of the inner state by the function units every time they are used, in order to prevent destruction of their functional capacity.

- Careful programming. This includes the clean use of data types and the maintenance of fixed conventions in the use and the avoidance of elements of programming languages (see [Schrem 74]).

/170

Very often additional internal system information is required for the analysis of a programming error. For each data structure in the project data bank and in the internal state of the program, there is a listing program in the system. Each of these programs can be activated on the command plane (by commands in the basic language). This makes it possible

for the user, after appropriate remarks (for example, from the error handbook), to list the surroundings of a programming error and make it available for the error analysis. In the vicinity of an error, the dynamic call sequence list of the subprograms is also important. For this purpose, one can have sequence traces which are easy to turn on and off which can be controlled from the command plane.

7.4 Data safety

In the model description, the safety of the data stored in the project data bank plays a central role. This is especially true when a project extends over a long period of time. The protection of the data against unauthorized access is an important requirement for model description. All of the comfortable operating systems provide extensive protective measures for a project data bank. Therefore, on the satellite system plane there do not have to be any corresponding measures for this. The following concepts of the satellite system are used to insure the safety of the model data in the sense of user support about the nature of the project data bank:

1. The user can operate the project data bank in the two access modes "active" and "passive" (see Chapter 5.1). One can easily change the operating mode. For many accesses, it is sufficient only to read the project data bank. Therefore, in these cases, the user himself can protect the project data bank against destruction.

/171

2. The project data bank is considered to be a generalized memory cell (Chapter 5.1). This means that the uniqueness of the data is insured. The data input last by the user will certainly be the actual data in the project data bank.

3. At any time the user can interrupt the work with the satellite system and take it up again at any later time ("break/restart"--capacity). Using the principle of generalized memory cells, the last data state is always conserved.

4. The system has an error tolerating behavior. An operating error in the use of the satellite system or a description error do not lead to a loss in project data in any case. For example, the first call of segments already consolidated is prevented.

5. According to the principle of reversibility of communications, the result of model description is also insured after a long connection work, because the production of a model data can always be done later on.

The protection of project data against destruction depends greatly on the reliability of the operating system and the computer (key words: "deadlock", "head-crash", power failure). The user can insure himself against this only by multiple and distributed storage of the project data (for example, on magnetic disks and magnetic tapes). However, there is no absolute data safety here.

Literatur

Ah72 AHO,A.V., ULLMAN,J.D., "The Theory of Parsing, Translation and Compiling", Prentice-Hall, Englewood Cliffs, 1972.

Ar54 ARGYRIS,J.H., "Energy Theorems and Structural Analysis", Butterworths, London, 1960 (ursprünglich veröffentlicht — in Aircraft Engng. 1954-1955).

Ar56 ARGYRIS,J.H., "The Matrix Analysis of Structures with Cut-Outs and Modifications", Communication to the IX. International Congress of Theoretical and Applied Mechanics (IUTAM), Brüssel, Sept. 1956, pp. 131-142.

Ar57 ARGYRIS,J.H., "Die Matrizentheorie der Statik", Ingenieur-Archiv 25 (1957), pp. 174-192.

Ar59 ARGYRIS,J.H., KELSEY,S., "Modern Fuselage Analysis and the Elastic Aircraft", Aircraft Engng 31 (1959), 33 (1961), als Buch : Butterworths, London, 1963.

Ar64 ARGYRIS,J.H., "Recent Advances in Matrix Methods of Structural Analysis", Progress in Aeronautical Sciences, Vol. 4, Pergamon Press, London, 1964.

Ar65 ARGYRIS,J.H., "Continua and Discontinua", in: Proc. of the Int. Conf. on Matrix Meth. of Struct. Mech., Wright-Patterson A.F.B., Dayton, Ohio (26. Okt. 1965), pp.1-198.

Ar68 ARGYRIS,J.H., FRIED,I., SCHARPF,D.W., "The TET20 and TEA8 Elements for the Matrix Displacement Method", Aeronaut. J. RAS 72 (1968), pp. 618-623.

Ar69 ARGYRIS,J.H., SCHARPF,D.W., "The Incompressible Lubrication Problem", Aeronaut. J. RAS 73 (1969), pp. 1044-1046.

Ar70 ARGYRIS,J.H., "The Impact of the Digital Computer on Engineering Sciences" (12th Lancaster Memorial Lecture), Aeronaut. J. RAS 74 (1970), pp. 13-41, 111-127.

Ar72 ARGYRIS,J.H., MARECZEK,G., "Potential Flow Analysis by Finite Elements", Ingenieur-Archiv 41 (1972), pp. 1-25.

Ar75 ARGYRIS,J.H., BRÖNLUND,O.E., "The Natural Factor Formulation of the Stiffness for the Matrix Displacement Method", Comp. Meth. Appl. Mech. Engng 5 (1975), pp. 97-119.

Ar82 ARGYRIS,J.H., "An Excursion Into Large Rotations", Comp. Meth. Appl. Mech. Engng 32 (1982), pp. 85-155.

AS71 ASKA Part I - Linear Static Analysis, User's Reference Manual, ISD-Report 73, Stuttgart, 1971, Revision F 1979, ASKA UM 202.

Ba76 BAUER,F.L., EICKEL,J. (Hg.), "Compiler Construction", Lecture Notes in Computer Science Bd. 21, Springer, Berlin-Heidelberg-New York, 1976 (2. Auflage).

Be75 BECKER,E., BÜRGER,W., "Kontinuumsmechanik", B.G. Teubner, Stuttgart, 1975.

Bö69 BÖHLING,K.H., INDERMARK,K., "Endliche Automaten", Bibliographisches Institut, Mannheim, 1969/70.

Br77 BRANIN,Jr.,F.H., HUSEYIN,K. (Hg.), "Problem Analysis in Science and Engineering", Interdisciplinary Conference, University of Waterloo, Ontario, Canada, Mai 1975, Academic Press, 1977.

Ch56 CHOMSKY,N., "Three Models for the Description of Language", I.R.E. Transactions on Information Theory, Vol. IT-2, Proc. of the Symp. on Inform. Theory, Sept. 1956.

Co76 COLE,A.J., "Macro Processors", Cambridge University Press, London, 1976.

Cs67 COONS,S.A., "Surfaces for Computer-Aided Design of Space-Forms", MIT-Project MAC, Report MAC-TR-41, 1967.

Da72 DAHL,O.-J., DIJKSTRA,E.W., HOARE,C.A.R., "Structured Programming", Academic Press, London, New York, 1972.

Di76 DIJKSTRA,E.W., "A Discipline of Programming", Prentice-Hall, Englewood Cliffs, 1976.

DIN1 DIN 1301, Einheiten (-namen, -zeichen), 1978/79.

DIN6 DIN 66001, Sinnbilder für Datenfluß- und Programmablaufpläne, 1969.

DIN7 DIN 66027, Programmiersprache FORTRAN, 1979.

Dö72 DÖRFLER,W., MÜHLBACHER,J., "Graphentheorie für Informatiker", Walter de Gruyter, Berlin und New York, 1973.

Fe64 FENVES,S.J., LOGCHER,R.D., MAUCH,S.P., REINSCHMIDT,K.F., "STRESS ; A User's Manual", MIT Press, 1964.

Ga77 GALLAGHER,R.H., "Computerized Structural Analysis and Design - The Next Twenty Years", Comp. and Structures 7 (1977), pp. 495-501.

Ge77 GEHANI,N., "Units of Measure as a Data Attribute", Computer Languages 2 (1977), pp. 93-111.

Gm75 GRAM,Chr., HERTWECK,F.R., "Command Languages : Design Considerations and Basic Concepts", in: UNGER,C. (Hg.), "Command Languages"(Proc. IFIP Congress on Command Languages), North-Holland, 1975.

Gr71 GRIES,D., "Compiler Construction for Digital Computers", Wiley International Edition, 1971.

Ho79 HORNING,J.J., "Programming Languages for Reliable Computer Systems", in: GOOS, HARTMANIS, BAUER, BROY (Hg.), "Program Construction", Lecture Notes in Comp. Science, Springer, 1979, pp. 494-530.

Iv77 IVIE,E.L., "The Programmer's Workbench - A Machine for Software Development", CACM 20 (1977), pp. 746-753.

Jo78 JONES,P.F., "Four Principles for Man-Computer Dialogue", CAD 10 (1978), pp. 197-202.

Kn68 KNUTH,D.E., "The Art of Computer Programming", Vol. 1 (Fundamental Algorithms), Addison-Wesley, Reading, 1968.

Kb81 KOLB,H., "Standardized Interfaces for Data Transfer and Graphics", in: ASKA UM 230, Stuttgart, 1981, pp. 84-98.

Ky79 KOWALSKY,H.-J., "Lineare Algebra", Walter de Gruyter, Berlin und New York, 1979 (9. Auflage).

Li79 LINGER,R.C., MILLS,H.D., WITT,B.I., "Structured Programming - Theory and Practice", Addison-Wesley, 1979.

Ma80 MACKENZIE,C.E., Coded Character Sets - History and Development", Addison-Wesley, 1980.

Me79 MELOSH,R.J., "Design Principles for Finite Element Meshing", NASA-ICASE Symposium on Math. Modelling, Langley Research Center, Hampton, 24. Okt. 1979.

C-3

Mü76 MÜLLER,K.P., WÖLPERT,H., "Anschauliche Topologie", B.G. Teubner, Stuttgart, 1976.

Pn77 PARNAS,D.L., "The Use of Precise Specifications in the Development of Software", in: GILCHRIST,B. (Hg.), "Information Processing 77" (Proc. IFIP Congress), North-Holland, 1977, pp. 861-867.

Ps79 PARSONS,I.T., "A Support System for Interactive Languages", Software 9 (1979), pp. 73-86.

Pf81 PFAFFINGER,D., "Zur automatischen FE-Netzgenerierung", International FEM-Congress (IKOSS), Baden-Baden, 1981.

Re81 REHAK,D.R., LOPEZ,L.A., "Computer Aided Engineering - Problems and Prospects", Technical Report of Research, Department of Civil Engineering, University of Illinois, Urbana, Illinois, Juli 1981.

Ri74 RITCHIE,D.M., "The UNIX Time-Sharing System", CACM 17 (1974), pp. 365-375.

Ro66 ROOS,D., "ICES System Design", MIT Press, 1966.

Rp79 ROPOHL,G., "Eine Systemtheorie der Technik - Zur Grundlegung der Allgemeinen Technologie", Hanser, München und Wien, 1979.

Rs70 ROSEN,R., RUBINSTEIN,M.F., "Substructure Analysis by Matrix Decomposition", J. Struct. Div. ASCE 96 (1970), pp. 663-670.

Ru76 RUTISHAUSER,H., "Vorlesungen über numerische Mathematik", Bd. 1 (Gleichungssysteme, Interpolation, Approximation), Birkhäuser, Basel und Stuttgart, 1976.

Sa73 SALOMAA,A., "Formal Languages", Academic Press, New York und London, 1973.

Sh82 SHNEIDERMAN,B., "Designing Computer System Messages", CACM 25 (1982), pp. 610-611.

Sm70a SCHREM,E., ROY,J.R., "An Automatic System for Kinematic Analysis ASKA Part I", in: FRAEIJS DE VEUBEKE,B. (Hg.), "High Speed Computing in Elastic Structures" (Proc. IUTAM-Symposium 1970), Universität Liège, 1970, pp. 447-507.

Sm70b SCHREM,E., "Die Konzipierung eines allgemeinen Rechenprogramms für die Anwendung der Methode der finiten Elemente", in: BUCK,K.E., SCHARPF,D.W., STEIN,E., WUNDERLICH,W. (Hg.), "Finite Elemente in der Statik" (DFG-Kolloquium Stuttgart 1970), Wilhelm Ernst und Sohn, München, 1973, pp. 302-320.

Sm71 SCHREM,E., "Computer Implementation of the Finite Element Procedure", in: FENVES,S.J., PERRONE,N., ROBINSON,A.R., SCHNOBRICH,W.C. (Hg.), "Numerical and Computer Methods in Structural Mechanics" (Proc. ONR-Conference 1971), Academic Press, 1973, pp. 79-121.

Sm74 SCHREM,E., "Standard FORTRAN IV (Erläuterungen und Ergänzungen)", ISD-Report 163, Stuttgart, 1974, Revision A, 1977.

Sm75 SCHREM,E., "A Short Description of ASKA", ISD-Report 194, Stuttgart, 1975.

Sm76 SCHREM,E., "PAGIO - A Software Package for Handling Paged Data-Sets", ISD-Report 198, Stuttgart, 1976.

Sm78a SCHREM,E., "Programmbausteine und Datenstrukturen für die Implementation der Methode der finiten Elemente", Dissertation, Universität Stuttgart, 1978.

Sm78b SCHREM,E., "Functional Software Design and its Graphical Representation", Comp. and Structures 8 (1978), pp. 491-502.

Sm79 SCHREM,E., SCHULZ,U., "PRELEX - Eine Funktionseinheit für die lexikalische Analyse", ISD-Bericht 265, Stuttgart, 1979.

Sm80 SCHREM,E., "Werkzeuge für die Modell-Beschreibung und Ergebnis-Darstellung in ASKA 80", International FEM-Congress (IKOSS), Baden-Baden, 17./18. Nov. 1980.

Su69 SCHUBERT,H., "Topologie", B.G. Teubner, Stuttgart, 1969 (2. Auflage).

Ta75 TAIG,I.C., "Modelling and Interpretation of Results in Finite Element Structural Analysis", in: ROBINSON and Ass. (Hg.), Proc. of the World Congr. on Finite Element Meth. in Struct. Mech., Bournemouth, Dorset, England, 12.-17. Okt. 1975, Vol. 1.

We71 WEINBERG,G.M., "The Psychology of Computer Programming", Van Nostrand Reinhold, 1971.

Wi71 WIRTH,N., "The Design of a PASCAL-Compiler", Software 1 (1971), pp. 309-333.

Wi82 WIRTH,N., "Programming in MODULA2", Springer, Berlin-Heidelberg-New York, 1982.

REFERENCES /172

Ar54 ARGYRIS, J. H., "Energy Theorems and Structural Analysis", Butterworths, London, 1960 (originally published in Aircraft Engineering, 1954-1955).

Ar57 ARGYRIS, J. H., "Matrix Theory of Statics", Ingenieur-Archiv 25 (1957), pp. 174-192.

Ba75 BAUER, F. L., EICKEL, J. (Hg.), "Compiler Construction", Lecture Notes in Computer Science Bd. 21, Springer, Berlin-Heidelberg-New York 1976 (2nd edition). /173

Be75 BECKER, E., BURGER, W., Continued Mechanics", B. G. Teubner, Stuttgart, 1975.

Bo69 BoeHLING, K.H., INDERMARK, K. Finite Automata, Bibliographisches Institut, Mannheim, 1969/70.

DIN1 DIN 1301, units (names, symbols), 1978/79.

DIN6 DIN 66001, Images for data flow and program sequence plans, 1969.

DIN7 DIN 66027, Programming language, FORTRAN, 1979.

Do72 DORFLER, W., MUHLBACHER, J., "Graph theory for information", Walter de Gruyter, Berlin and New York, 1973.

/175

Mu76 MULLER, K. P., WOLPERT, H., "Lucid Topology", B. G. Teubner, Stuttgart, 1976.

Pf81 PFAFFINGER, D., "Automatic FE Network Generation", International FEM-Congress (IKOSS), Baden-Baden, 1981.

Rp79 ROPOHL, G., "A System Theory of Technology-- Foundations of General Technology", Hanser, Munich and Vienna, 1979.

Ru76 RUTISHAUSER, H., "Lectures on Numerical Mathematics", Vol, 1 (Equation Systems, Interpolation, Approximation), Birkhauser, Basel and Stuttgart, 1976.

Sm70b SCHREM, E., "The Concept of a General Computer Program for the Use of the Finite Element Method". In: BUCK, K. E., SCHARPF, D. W., STEIN, E., WUNDERLICH, W. (Hg.), "Finite Elements in Statics" (DFG-Colloquium Stuttgart 1970, Wilhelm Ernest and Sohn, Munich, 1973, pp. 302-320. /176

Sm78a SCHREM, E., "Program Blocks and Data Structures for the Implementation of the Finite Element Method", Dissertation, University of Stuttgart, 1978.

SM79 SCHREM, E., SCHULZ, U., "PRELEX--A Functional Unit for Lexical Analysis", ISD Report 265, Stuttgart, 1979.

Sm80 SCHREM, E., "Tools for Model Description and Result--Presentation in ASKA 80", International FEM-Congress (IKOSS), Baden Baden, 17.18. Nov. 1980.

Su69 SCHUBERT, H., "Topology", B. G. Teubner, Stuttgart, 1969 (2nd Edition).

FIGURE LIST

Figure /178

3.1 Examples of mesh types from model description in structural calculation

3.2 Figure of a mesh tree

3.3 Compatibility of mesh types in the lower mesh/upper mesh relationship

3.4 Commands for mesh tree description (for example, see Figure 4.5)

3.5 Degress in incidents for topological units in finite element meshes

3.6 Characteristics of elements

3.7 Commands for describing topology

3.8 Simple example for topology description

3.9 Correspondence of mesh coordinates and world coordinates

3.10 Coordinate systems and bases for describing configuration

3.11 Association of mesh and world node bases

3.12 Determination methods for node bases

3.13 Commands for describing configuration

3.14 Simple examples for describing configuration

3.15 Relationships between pairs of dual quantities

3.16 The duality ladder for the displacement method in structural calculations

3.17 Division of the degrees of freedom

3.18 Degrees of freedom distribution plan of a partial mesh

3.19 Commands for describing kinematic boundary conditions

3.20 Examples for describing kinematic boundary conditions /179

3.21 Commands for describing elementary data

3.22 Commands for describing loads

3.23 Simple examples for describing node loads

4.1 User sequence diagram of a section

4.2 State diagram of a section

4.3 State diagram for data access using contexts

4.4 State diagram for data access using contexts and olymp

4.5 Examples for representing a mesh tree as a mesh list and its description

4.6 General program sequence plan for mesh tree crossing

4.7 State diagram of the net context control

5.1 State diagram of sequential management

5.2 Commands for system description

5.3 Model data and their data modules

5.4 State diagram of table management

5.5 Commands for project monitoring

6.1 The command language structures the user interface

6.2 Left linear production system for the tokens of the command language

6.3 Project system for the class of command languages

6.4 The recognition operator $(s_i \xrightarrow{v_j} s_j)$

6.5 Two representations of the syntactic structures of the command language /180

6.6 Self description of the command language

6.7 The token types and their values (subtypes)

6.8 Hierarchy of the syntactic units of command languages

6.9 Summary diagrams of the components of the language generator

6.10 Representation of token types and variables in the data register

6.11 Calling sequence diagram of functions of the language translator

6.12 Calling sequence diagram of functions of the language translator with consideration of transparent command

6.13 Syntax of open list

6.14 Calling sequence diagram of functions of the language translator and the listing generator

6.15 Schematic program flow chart of the language processing

6.16 Syntax diagram of the description of number sequences

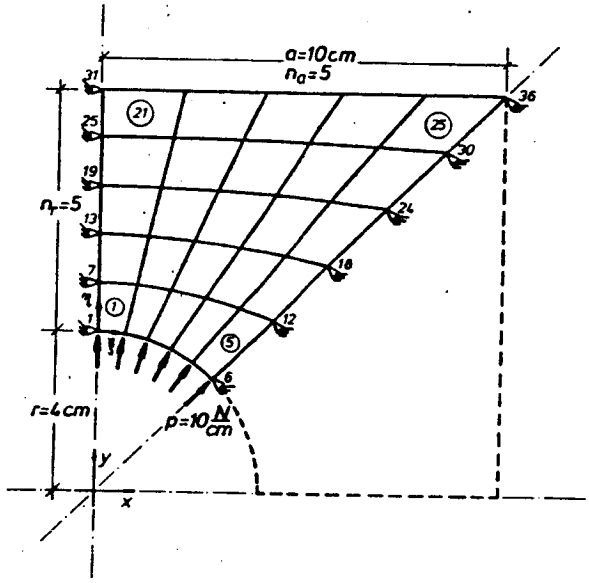
6.17 Examples for numerical sequences of the first and second degrees

6.18 Phases of command processing

6.19 Syntax diagram of macro description

Appendix A: Example for model description

We wish to describe a square plate with a hole, whose hole edge is loaded with the pressure p. Based on the symmetry of the hole plate and the load, the idealization of one-eighth of the hole plate is sufficient. This segment has to be handled kinematically in such a way that the displacement perpendicular to the symmetry lines is not possible.



The dimensions of the model are as follows:

half edge length of the square	a	= 10 cm
whole radius	r	= 4 cm
number of elements in the edge direction	n_a	= 5
number of elements in the radial direction	n_r	= 5
pressure at the hole edge	p	= 10 N/cm
elasticity modulus of the plate material	E	=100 N/cm
Poisson transverse contraction coefficient	ν	= 0.3

The configuration of the mesh is determined by a two /182
dimensional continuum. Correspondingly, a continuum element QUAC4 with 4 nodes and the degrees of freedom u and v at each node is used as the element. We thus obtain the

following model description (see Chapter 3), where the key words are written in large letters and the identifiers are written in small letters for readability:

The mesh tree is of the simplest kind. An element mesh (1) is coupled into a composite mesh (10), which is also the main mesh.

```
TREE NEW
MAIN 10 NODES 36 CASES 1 NETYP pc SUB 1
ENET  1 ELTS  25         ELTYP quac4 MOTYP isot LATYP no
END
```

Starting with the ordinal node and elementary numbering, the element indices are first described in the circumferential direction (first loop: LOOP) and in the radial direction (second loop: AND).

```
TOPO NEW
NODESEQ 10 NODES ORDINAL
ELTSEQ   1 ELTS  ORDINAL
INSELT   1 ALL ELTS FROM 1,2,8,7 LOOP 5 BY 1,1,1,1 AND 5 BY 6,6,6,6 REP
END
```

The node coordinates are described in three steps:

1. The coordinates along the inner edge are determined with polar coordinates ("polar").

2. The coordinates along the outer edge are determined with cartesian coordinates "SEL". In the mesh tree description, the mesh type "pc" was given (see Figure 3.1). In this way, the cartesian coordinate system is specified as a network coordinate system and all node positions are converted into this system.

3. The other node coordinates are interpolated through the topological rectangle range 1, 6, 36, 31 ("PARC"). For this purpose, along the inner edge

a trial solution of a fifth order (6 support points) and a linear trial solution along the other edges (2 support points) is used. In the circumferential direction the ξ-parameters are given and the η-parameters are given in the radial direction.

The node bases "ROTB" are described so that the u-degree of freedom along the left and right edge ("REF y") and the v-degree of freedom along the inner and outer edge ("LB REF y") are perpendicular to the edge. The other node bases follow in the directions of the mesh base.

```
CONFIG NEW
COOR 10 polar rho(cm). phi(grad) TAB FROM 1,4,90 LOOP 6 TO 6,4,45 REP
        SEL   x  (cm) y  (cm)   TAB FROM 31,0,10 LOOP 6 TO 36,10,10 REP
PARC 10 XI 6,2 ETA 2 BOUND RECTA FROM 1 LOOP 6 BY 1 REP,36,31
        ALL FROM 0,0 LOOP 6 BY 0.2,0 AND 6 BY 0,0.2 REP
ROTB 10     REF y TAB FROM 6,6,36 LOOP 6 BY 6,0,0 REP
        LB REF y TAB FROM 1,1 LOOP 6 BY 1,1 REP
END
```

All the degrees of freedom along the edge which are perpendicular to the symmetry lines of the whole plate are suppressed ("supp u"). In these directions, there can be no displacement. The alignment of the degress of freedom is specified by node bases.

```
BOUND NEW
FRED 10 supp u IN FROM 1 LOOP 6 BY 6 AND 2 BY 5 REP
END
```

The elasticity modulus ("emod") and the transverse contraction coefficient ("ny") for all elements are given for element data at the same level.

```
ELDA NEW
ELDA 1 emod (N cm -1) WITH 100   ELTS ALL
       ny             WITH   0.3 ELTS ALL
END
```

Only a single load case is considered ("LCASEQ", "LCASE"). This consists of pressure forces of the same magnitude along the nodes of the inner edge ("NLOAD"). /184

The radial direction occurs there for the v-degree of freedom, which is aligned by the node base accordingly.

```
LOAD NEW
LCASEQ 1
LCASE 1
NLOAD 10 SEL v(N) TAB FROM 1,10 LOOP 6 BY 1,0 REP
END
```

When one wishes to calculate a sequence of such models with different dimensions, the use of macro commands for model description is very advantageous for such a parameter investigation. The commands given above are then generalized as follows:

1. The macro command "plate" is taken over by the parameters of the model and it controls the model description in the individual segments.

2. All commands for one segment are included in a macro command. The required commands for macro description are discussed in Chapter 6.2.2.

3. Each such macro command is supplied with a number of model parameters. The corresponding variables are called GLO and are defined as being dependent on the surroundings. In addition, a few variables independent of the surroundings are required. They are called LOC.

4. Certain fixed values in the commands will also have to be changed for a changed model. These new values are now in general calculated using the LET command from the model parameters and assigned to certain local variables. The variables are set between $-signs and instead of the fixed values they are introduced into the command.

/185

```
DEFMACRO plate TO file
   DECLARE
      ARG NUM a  SUB 4 - 5
      ARG NUM r  SUB 4 - 5
      ARG NUM na SUB 4 - 4
      ARG NUM nr SUB 4 - 4
      ARG NUM p
      ARG NUM e  SUB 4 - 5
      ARG NUM ny SUB 4 - 5
   ENDDEC
   BODY
      TREE NEW
      MACRO treeplate FROM file START
      TOPO NEW
      MACRO topoplate FROM file START
      CONFIG NEW
      MACRO configplate FROM file START
      BOUND NEW
      MACRO boundplate FROM file START
      ELDA NEW
      MACRO eldaplate FROM file START
      LOAD NEW
      MACRO loadplate FROM file START
   ENDBODY
ENDDEF

DEFMACRO treeplate TO file
   DECLARE
      GLO NUM na
      GLO NUM nr
      LOC NUM nodes
      LOC NUM elts
   ENDDEC
   BODY
         LET elts EQU na MPY nr
         LET nodes EQU elts ADD na ADD nr ADD 1
      MAIN 1O NODES $nodes$ CASES 1 NETYP pc SUB 1
      ENET  1 ELTS  $elts$          ELTYP quac4 MOTYP isot LATYP no
      END
   ENDBODY
ENDDEF

DEFMACRO topoplate TO file
   DECLARE
      GLO NUM na
      GLO NUM nr
      LOC NUM n1
      LOC NUM n2
      LOC NUM n3
   ENDDEC
   BODY
         LET n1 EQU na ADD 1
         LET n2 EQU na ADD 2
         LET n3 EQU na ADD 3
      NODESEQ 1O NODES ORDINAL
      ELTSEQ   1 ELTS  ORDINAL
      INSELT   1 ALL ELTS FROM 1,2,$n3$,$n2$ LOOP $na$ BY 1,1,1,1
                          AND $nr$ BY $n1$,$n1$,$n1$,$n1$ REP
      END
   ENDBODY
ENDDEF
```

```
DEFMACRO configplate TO file
   DECLARE
      GLO NUM a
      GLO NUM r
      GLO NUM na
      GLO NUM nr
      LOC NUM n1
      LOC NUM n2
      LOC NUM n3
      LOC NUM t1
      LOC NUM t2
   ENDDEC
   BODY
         LET n1 EQU na ADD 1
         LET n2 EQU n1 MPY nr ADD 1
         LET n3 EQU n2 ADD na
      COOR 10 polar rho(cm) phi(grad) TAB FROM 1,$r$,90 LOOP $n1$
                                          TO $n1$,$r$,45 REP
              SEL   x  (cm) y  (cm)   TAB FROM $n2$,0,$a$ LOOP $n1$
                                          TO $n3$,$a$,$a$ REP
         LET t1 EQU 1 DIV na
         LET t2 EQU 1 DIV nr
         LET n2 EQU nr ADD 1
      PARC 10 XI $n1$,2 ETA 2 BOUND RECTA FROM 1 LOOP $n1$ BY 1 REP,$n3$,$n2$
              ALL FROM 0,0 LOOP $n1$ BY $t1$,0 AND $n2$ BY 0,$t2$ REP
      ROTB 10    REF y TAB FROM $n1$,$n1$,$n3$ LOOP $n2$ BY $n1$,0,0 REP
              LB REF y TAB FROM 1,1 LOOP $n1$ BY 1,1 REP
      END
   ENDBODY
ENDDEF

DEFMACRO boundplate TO file
   DECLARE
      GLO NUM na
      GLO NUM nr
      LOC NUM n1
      LOC NUM n2
   ENDDEC
   BODY
         LET n1 EQU nr ADD 1
         LET n2 EQU na ADD 1
      FRED 10 supp u IN FROM 1 LOOP $n1$ BY $n2$ AND 2 BY $na$ REP
      END
   ENDBODY
ENDDEF

DEFMACRO eldaplate TO file
   DECLARE
      GLO NUM e
      GLO NUM ny
   ENDDEC
   BODY
      ELDA 1 emod (N cm -1) WITH $e$  ELTS ALL
             ny             WITH $ny$ ELTS ALL
      END
   ENDBODY
ENDDEF
```

/187

```
DEFMACRO loadplate TO file
   DECLARE
      GLO NUM na
      GLO NUM p
      LOC NUM n1
   ENDDEC
   BODY
         LET n1 EQU na ADD 1
      LCASEQ 1
      LCASE 1
      NLOAD 10 SEL v(N) TAB FROM 1,$p$ LOOP $n1$ BY 1,0 REP
      END
   ENDBODY
ENDDEF
```

The model given above is described by the macro command

```
MACRO plate FROM file NUM a  =  10
                      NUM r  =   4
                      NUM na =   5
                      NUM nr =   5
                      NUM p  =  10
                      NUM e  = 100
                      NUM ny =   0.3 START
```

By changing the arguments, one can produce other idealizations of the plate.

/188

Appendix B: Linear static analysis

In the following we will give all of the calculation steps required for the meshes of the mesh tree in the case of a linear static analysis. The method is used which is implemented in the programming system ASKA [ASKA 71].

We will first consider a partial network with n submeshes. For each submesh j ($1 \leq j \leq n$) we assume the existence of class specific degrees of freedom index matrices a_{Uj}, a_{Ej}, a_{Pj}, a_{Sj} and for the partial mesh the class selection matrices b_U, b_E, b_P, b_S. In addition, for each submesh, we assume a stiffness matrix k_j and a matrix of the node forces Q_j for the degrees of freedom coupled to the partial mesh. These matrices are determined for the elementary meshes by the element calculation (Chapter 3.11). Therefore, the FE (finite element) calculation starts in the partial meshes into which only elementary meshes have been coupled.

The assembly in the partial mesh gives the following for the node forces

$$R_U = R'_U + \sum_{j=1}^{n} a_{Uj}^T Q_j$$
$$R_E = R'_E + \sum_{j=1}^{n} a_{Ej}^T Q_j \qquad \text{(B.1)}$$

etc.,

where

$$R'_U = b_U^T R'$$
$$R'_E = b_E^T R' \qquad \text{(B.2)}$$

etc.

R' contains the known node forces input in the partial mesh. For the assembled stiffnesses, we find

$$K_{UU} = \sum_{j=1}^{n} a_{Uj}^T k_j a_{Uj} \qquad \text{(B.3)}$$
$$K_{UE} = \sum_{j=1}^{n} a_{Uj}^T k_j a_{Ej}$$

etc.

/189

In this way one obtains the linear equation system for the constitution equation

$$\begin{bmatrix} R_U \\ R_E \\ R_P \\ R_S \end{bmatrix} = \begin{bmatrix} K_{UU} & K_{UE} & K_{UP} & K_{US} \\ & K_{EE} & K_{EP} & K_{ES} \\ & & K_{PP} & K_{PS} \\ & \text{sym.} & & K_{SS} \end{bmatrix} \begin{bmatrix} r_U \\ r_E \\ r_P \\ r_S \end{bmatrix} . \qquad \text{(B.4)}$$

If we consider that

$$r_S = 0$$

and if we eliminate the prescribed displacement

$$r_P = b_P^T r'_P \qquad \text{(B.5)}$$

with the displacements r'_p input in the partial mesh, then one obtains the equation system

$$\begin{bmatrix} \bar{R}_U \\ \bar{R}_E \end{bmatrix} = \begin{bmatrix} K_{UU} & K_{UE} \\ K_{UE} & K_{EE} \end{bmatrix} \cdot \begin{bmatrix} r_U \\ r_E \end{bmatrix} \tag{B.6}$$

where

$$\bar{R}_U = R_U - K_{UP} r_P \tag{B.7}$$
$$\bar{R}_E = R_E - K_{EP} r_P$$

The triangulation of K_{UU} by Cholesky gives

$$K_{UU} = U^T U \tag{B.8}$$

Using

$$\bar{r}_U = U^{-T} \bar{R}_U \tag{B.9}$$

and

$$T_{UE} = U^{-T} K_{UE} \tag{B.10}$$

one obtains the coupling matrices of the partial mesh /190

$$k = K_{EE} - T_{UE}^T T_{UE}$$
$$Q = \bar{R}_E - T_{UE}^T \bar{r}_U \tag{B.11}$$

If the degree of freedom index matrices

$$a_U, a_E, a_P, a_S$$

unknown in the partial mesh, then the calculation on the upper mesh plane can be continued with the same method. This process is repeated until the main mesh is reached. There we have

$$r_E = 0$$

and from equation (B.6), we find

$$\bar{R}_U = K_{UU} r_U \tag{B.12}$$

Using equations (B.8) and (B.9), we find the solution

$$r_U = U^{-1} \bar{r}_U \tag{B.13}$$

In the following calculation, we find the following in each submesh j ($r_S = 0$)

$$r_{Ej} = \begin{bmatrix} a_{Uj} & a_{Ej} & a_{Pj} \end{bmatrix} \begin{bmatrix} r_U \\ r_E \\ r_P \end{bmatrix} \tag{B.14}$$

and one finds with equation (B.9) and (B.10) from (B.6) that

$$r_{Uj} = U^{-1}(\bar{r}_{Uj} - T_{UE} r_{Ej}) \qquad \text{(B.15)}$$

This process of reverse calculation of the displacements is continued over each of the submeshes until all elementary meshes have been reached and the element displacements are calculated.

/191

Appendix C: Proof of equation (3.15)

$$f^{i,j}_{(n,m)_{1,2}}(\xi,\eta) = f^{i}_{n_J}(\xi) f^{\bar{J}}_{1}(\eta) + f^{j}_{m_I}(\eta) f^{\bar{I}}_{1}(\xi) - f^{\bar{I}}_{1}(\xi) f^{\bar{J}}_{1}(\eta) \qquad \text{(3.15)}$$

Two conditions have to be satisfied for the correct sets of interpolation functions:

1. The function belong to a support node has the value of 1 at this node and at all other support nodes, it has the value 0.

   This condition is satisfied by definition for LaGrange polynomials (see equation (3.9)). This therefore is also true for the product of two such polynomials, if one is taken in ξ and the other η. The product becomes one at the support points, where both polynomials are 1 and are 0 otherwise. Since only support nodes are selected along the edge of the topological region, in equation (4.15) only the first two terms of the right side are equal to 1 for the corner nodes. This then means that the third term is 1 and the sum of all terms is also 1. For the other nodes, only one of the two other first terms is 1. Then, however, the third term is 0 and the sum of all terms is again 1. q.e.d.

2. A given set of interpolation functions have to be capable of interpolating a constant value over the corresponding

topological region. Then the sum of the interpolation functions has to be 1.

The sum of all functions according to equation (3.15) is the following for one side. For example, j = 1:

$$\sum_{i=1}^{n+1} f^{i,j}_{(n,m)_{1,2}}(\xi,\eta) = (\sum_{i=1}^{n+1} f^{i}_{n_1}(\xi)) f^{1}_{1}(\eta)$$
$$+f^{1}_{m_1}(\eta) f^{1}_{1}(\xi)+f^{1}_{m_2}(\eta) f^{2}_{1}(\xi)$$
$$-f^{1}_{1}(\xi) f^{1}_{1}(\eta)-f^{2}_{1}(\xi) f^{1}_{1}(\eta)$$

/192

For

$$\sum_{i=1}^{n+1} f^{i}_{n_1}(\xi) = 1$$

and

$$f^{1}_{1}(\xi) + f^{2}_{1}(\xi) = 1$$

we find

$$\sum_{i=1}^{n+1} f^{i,j}_{(n,m)_{1,2}}(\xi,\eta) = f^{1}_{m_1}(\eta) f^{1}_{1}(\xi) + f^{1}_{m_2}(\eta) f^{2}_{1}(\xi)$$

For the sum of all of the four sides of the region, we find

$$\sum_{j=1}^{m+1}\sum_{i=1}^{n+1} f^{i,j}_{(n,m)_{1,2}}(\xi,\eta) = f^{1}_{m_1}(\eta) f^{1}_{1}(\xi)+f^{1}_{m_2}(\eta) f^{2}_{1}(\xi)+$$
$$f^{m+1}_{m_1}(\eta) f^{1}_{1}(\xi)+f^{m+1}_{m_2}(\eta) f^{2}_{1}(\xi)+$$
$$f^{1}_{n_1}(\xi) f^{1}_{1}(\eta)+f^{1}_{n_2}(\xi) f^{2}_{1}(\eta)+$$
$$f^{n+1}_{n_1}(\xi) f^{1}_{1}(\eta)+f^{n+1}_{n_2}(\xi) f^{2}_{1}(\eta) \quad . \qquad \text{(C.1)}$$

For the sum of all four corner functions, we find:

$$
\begin{aligned}
f^{1,1}_{n_1,m_1}(\xi,\eta)+f^{n+1,1}_{n_1,m_2}(\xi,\eta)+f^{1,m+1}_{n_2,m_1}(\xi,\eta)+f^{n+1,m+1}_{n_2,m_2}(\xi,\eta) = \\
f^1_{n_1}(\xi)f^1_1(\eta)+f^1_{m_1}(\eta)f^1_1(\xi)-f^1_1(\xi)f^1_1(\eta)+ \\
f^1_{n_2}(\xi)f^2_1(\eta)+f^{m+1}_{m_1}(\eta)f^1_1(\xi)-f^1_1(\xi)f^2_1(\eta)+ \\
f^{n+1}_{n_2}(\xi)f^2_1(\eta)+f^{m+1}_{m_2}(\eta)f^2_1(\xi)-f^2_1(\xi)f^2_1(\eta)+ \\
f^{n+1}_{n_1}(\xi)f^1_1(\eta)+f^1_{m_2}(\eta)f^2_1(\xi)-f^2_1(\xi)f^1_1(\eta)
\end{aligned}
\qquad (C.2)
$$

Since

$$
\begin{aligned}
f^1_1(\xi,f^1_1(\eta)+f^1_1(\xi)f^2_1(\eta)+f^2_1(\xi)f^2_1(\eta)+f^2_1(\xi)f^1_1(\eta) = \\
f^1_1(\xi)\left[f^1_1(\eta)+f^2_1(\eta)\right]+f^2_1(\xi)\left[f^2_1(\eta)+f^1_1(\eta)\right] = 1
\end{aligned}
$$

one obtains the value of 1 for the difference between equations (C.1) and (C.2). q.e.d.

HISTORY /193

Personal: Wilfried Reinhard Helfrich, born Wagele, born [redacted] in [redacted], married to Brigitte Helfrich since 1978.

School: 1960-1964 grade school
1964-1972 gymnasium
both in Ludwigsburg

Matura: 1972 at the Friedrich-Schiller-Gymnasium in Ludwigsburg

Studies: After winter semester 1973/74 at the Stuttgart University, Division for Aerodynamics and Space Flight.
One-half year practical activity at various firms.

1979 main diploma examination

Scientific activity:

After my studies, scientific worker at the Institute for Statics and Dynamics of the Aviation and Space Flight Division, Stuttgart University.

STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-88528		

4. Title and Subtitle	5. Report Date
DESIGN OF SOFTWARE FOR DESIGN OF FINITE ELEMENT FOR STRUCTURAL ANALYSIS	February 1987
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Reinhard Helfrich	
	10. Work Unit No.
9. Performing Organization Name and Address	11. Contract or Grant No.
SCITRAN Box 5456 Santa Barbara, CA 93108	NASw-4004
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Translation
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

Translation of: "Zur Entwicklung eines Softwaresystems fur die Modellbeschreibung bei der Methode der finiten Elemente," dissertation submitted to the Faculty of Aerospace Transportation Technology of the University of Stuttgart for the Achievement of the Degree of Doctor of Engineering, (Nov. 22) 1983, pp. 1-192

16. Abstract

This work is concerned with the concepts of software engineering which allow a user of the finite element method to describe his model, to collect and to check the model data in a data base as well as to form the matrices required for a finite element calculation. Next the componenets of the model description are conceived including the mesh tree, the topology, the configuration, the kinematic boundary conditions, the data for each element and the loads. For this the possibilities for description and review of the data are especially considered. The concept of the segments for the modularization of the programs follows the components of the model description. The significance of the mesh tree as a as a global guiding structure will be understood in view of the principle of the unity of the model, mesh tree, and database...the user-friendly aspects of the software system will be summarized: the principle of language communication, the data generators, error processing, and data security.

17. Key Words (Selected by Author(s))	18. Distribution Statement
	Unclassified and Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	215	